IHE Structured Data Capture (SDC)

Technical Reference Guide (TRG)

Contents

[1 Introduction to Structured Data Capture (SDC) 6](#_Toc535360598)

[1.1 Prerequisites (TBD) 6](#_Toc535360599)

[1.2 The Lives of Data (TBD) 6](#_Toc535360600)

[1.3 What is Structured Data Capture (SDC)? 6](#_Toc535360601)

[1.4 SDC’s History and Objectives 6](#_Toc535360602)

[2 Overview of SDC Principles 7](#_Toc535360603)

[2.1 SDC Design Principles 7](#_Toc535360604)

[2.2 SDC Actors 8](#_Toc535360605)

[2.3 The SDC Information Model: Data Entry Forms and Data Elements 9](#_Toc535360606)

[2.3.1 Data Elements 9](#_Toc535360607)

[2.3.2 Mapping Data Elements to SDC forms 10](#_Toc535360608)

[2.3.3 The SDC Schema 11](#_Toc535360609)

[3 Data Entry Forms 11](#_Toc535360610)

[3.1 Introduction to Data Entry Forms 11](#_Toc535360611)

[3.2 Reporting from DEFs 12](#_Toc535360612)

[4 The Form Design File (FDF) 13](#_Toc535360613)

[4.1 Some Important Additional Definitions for SDC FDFs 13](#_Toc535360614)

[4.2 FDF Structural Overview 14](#_Toc535360615)

[4.2.1 SDC Conventions 14](#_Toc535360616)

[4.2.2 A First Look at FDF XML: 15](#_Toc535360617)

[4.2.3 The XFCs 17](#_Toc535360618)

[4.3 FormDesign Attributes and Properties 20](#_Toc535360619)

[4.3.1 FDF Namespaces 21](#_Toc535360620)

[4.3.2 FormDesign Attributes 21](#_Toc535360621)

[4.3.3 FDF Identifiers 23](#_Toc535360622)

[4.3.4 FormDesign Properties (eCC) 24](#_Toc535360623)

[5 Introduction to SDC Basic Schema Types 26](#_Toc535360624)

[5.1 SDC Schema File Overview 26](#_Toc535360625)

[5.2 Schema files. 26](#_Toc535360626)

[5.2.1 Basic SDC Schema Type Hierarchy 26](#_Toc535360627)

[5.3 The SDC Schema Inheritance Model and XFC Definitions 28](#_Toc535360628)

[5.4 The BaseType (abstract) 31](#_Toc535360629)

[5.5 The ExtensionBaseType (abstract) 32](#_Toc535360630)

[5.5.1 The EBT Property Element 32](#_Toc535360631)

[5.5.2 **FormDesign\Property** elements 33](#_Toc535360632)

[5.5.3 The EBT Comment Element 37](#_Toc535360633)

[5.5.4 The EBT Extension Element 37](#_Toc535360634)

[5.6 IdentifiedExtensionType (abstract): 37](#_Toc535360635)

[5.7 DisplayedType 37](#_Toc535360636)

[5.8 RepeatingType 37](#_Toc535360637)

[6 The XFCs 38](#_Toc535360638)

[6.1 XFC Identifiers and Names 38](#_Toc535360639)

[6.2 The DisplayedItem XFC 38](#_Toc535360640)

[6.2.1 DI Substructure 38](#_Toc535360641)

[6.3 The Section XFC 40](#_Toc535360642)

[6.4 The Question XFC 40](#_Toc535360643)

[6.4.1 Question-Response (QR) 40](#_Toc535360644)

[6.4.2 Single Select Questions 42](#_Toc535360645)

[6.4.3 Multi-Select Questions 43](#_Toc535360646)

[6.4.4 Capturing User Responses in Questions 43](#_Toc535360647)

[6.4.5 Response Metadata 44](#_Toc535360648)

[6.5 DEF Helper Components 44](#_Toc535360649)

[7 DEF Functional Considerations 45](#_Toc535360650)

[7.1 The DEF Maintains and Manipulates the FDF 45](#_Toc535360651)

[7.2 Implied Activation (IA) 45](#_Toc535360652)

[7.2.1 Implied Activation with Complex Nesting and Invisible Sub-Questions 46](#_Toc535360653)

[7.2.2 Complex Item Dependencies 47](#_Toc535360654)

[7.3 Invisible Question Text 47](#_Toc535360655)

[7.4 The @mustImplement Attribute 48](#_Toc535360656)

[7.4.1 Optional ListItems 48](#_Toc535360657)

[7.5 Required Responses 48](#_Toc535360658)

[7.6 Conditionally Required (CR) Behaviors and Reporting from the DEF 49](#_Toc535360659)

[7.6.1 Use of the “?” Prefix Display Model: 50](#_Toc535360660)

[7.6.2 Omit When Unanswered (OWU) 50](#_Toc535360661)

[7.6.3 Omit When Selected (OWS) 50](#_Toc535360662)

[7.7 Contiguity of ListItem Lists 52](#_Toc535360663)

[7.8 The Null Check Box 53](#_Toc535360664)

[7.9 The Single Check Box 53](#_Toc535360665)

[7.9.1 Reporting from the Single Check Box 53](#_Toc535360666)

[7.9.2 The Locked QAS 53](#_Toc535360667)

[7.10 Flavors of Unanswerable (FOU) 54](#_Toc535360668)

[8 Rules 55](#_Toc535360669)

[8.1 Selection Disables Children 55](#_Toc535360670)

[8.2 Selection Deselects Siblings 56](#_Toc535360671)

[8.3 Question Rules 57](#_Toc535360672)

[8.4 ListItem Rules 57](#_Toc535360673)

[8.5 Events and Guards 57](#_Toc535360674)

[8.5.1 Form Events 57](#_Toc535360675)

[8.5.2 DisplayedItem Events and Guards 57](#_Toc535360676)

[8.5.3 Question Events and Guards 57](#_Toc535360677)

[8.5.4 ListItem Events and Guards 57](#_Toc535360678)

[8.6 Form-Level (Polling) Rules 57](#_Toc535360679)

[8.6.1 Validation Rules 57](#_Toc535360680)

[8.6.2 Auto-Activation 57](#_Toc535360681)

[8.6.3 Auto-Selection 57](#_Toc535360682)

[8.6.4 Scripted Rules 57](#_Toc535360683)

[8.6.5 Call Rule 57](#_Toc535360684)

[8.6.6 External Rules (Web Services) 57](#_Toc535360685)

[8.6.7 ConditionalGroupActions 57](#_Toc535360686)

[8.7 Actions 57](#_Toc535360687)

[8.7.1 Action Types 57](#_Toc535360688)

[8.7.2 ConditionalGroupActions 57](#_Toc535360689)

[9 Repeating Sections and Questions 58](#_Toc535360690)

[9.1.1 Managing Repeated XFC and Component IDs with Monotonically Increasing Suffixes 58](#_Toc535360691)

[9.1.2 Nested Repeats 60](#_Toc535360692)

[10 DEF Validation and Reporting Results 63](#_Toc535360693)

[10.1 Incomplete/Invalid DEF 63](#_Toc535360694)

[10.2 Validating, Saving and Reporting from the DEF 63](#_Toc535360695)

[10.3 Validation and Reporting Test Template 63](#_Toc535360696)

[11 Generating Reports from a DEF 64](#_Toc535360697)

[12 SDC Instance Metadata 65](#_Toc535360698)

[13 The SDCPackage and its Metadata 65](#_Toc535360699)

[13.1 The SDC Package 65](#_Toc535360700)

[13.2 Package Attributes 66](#_Toc535360701)

[13.3 The Admin Element 66](#_Toc535360702)

[13.4 Package Nesting 66](#_Toc535360703)

[13.5 SubmissionRule 66](#_Toc535360704)

[13.6 ComplianceRule 66](#_Toc535360705)

[13.7 SDC HTML, XML and URL Options 66](#_Toc535360706)

[13.7.1 DemogFormDesign 66](#_Toc535360707)

[13.8 SDC Maps 66](#_Toc535360708)

[14 Pre-population of SDC Forms 67](#_Toc535360709)

[14.1 Form Pre-population (Pre-pop) and Auto-population (Auto-pop) 67](#_Toc535360710)

[14.2 Pre-pop 67](#_Toc535360711)

[14.3 Auto-pop 67](#_Toc535360712)

[14.4 Role and Structure of SDC Maps 67](#_Toc535360713)

[15 Submitting SDC Forms to FormReceivers 67](#_Toc535360714)

[15.1 Instance Metadata for SDC Package and Form Submissions 67](#_Toc535360715)

[16 The IHE SDC REST API 67](#_Toc535360716)

[16.1 Empty SDC Forms 67](#_Toc535360717)

[16.2 SDC Instance Forms 67](#_Toc535360718)

[17 Deprecated Content 67](#_Toc535360719)

[17.1 When are Items Deprecated? 67](#_Toc535360720)

[17.1.1 Question deprecation occurs when: 68](#_Toc535360721)

[17.1.2 Question deprecation does not occur when the following isolated changes occur: 68](#_Toc535360722)

[17.1.3 ListItems are deprecated when the following isolated changes to the ListItem occur: 68](#_Toc535360723)

[17.1.4 ListItem deprecation generally does not occur when the following isolated changes occur: 68](#_Toc535360724)

[18 Versioning 70](#_Toc535360725)

[18.1 FDF Versioning 70](#_Toc535360726)

[18.2 Package Versioning 70](#_Toc535360727)

[18.3 eCC Versioning 70](#_Toc535360728)

[18.3.1 The eCC FDF Version: 70](#_Toc535360729)

[18.3.2 Relationship of FDF Versions to CCP Versions 71](#_Toc535360730)

[19 SDC Security Considerations 72](#_Toc535360731)

[20 The Basic XML Structure of FDF Files (old) 72](#_Toc535360732)

[20.1 Structure of FDF Elements and Attributes for Template Line Items: 72](#_Toc535360733)

[20.2 Metadata in <template-header > 73](#_Toc535360734)

[20.2.1 Identifiers 73](#_Toc535360735)

[20.2.2 CCP Metadata in the FDF file 73](#_Toc535360736)

[20.2.3 Other Metadata 73](#_Toc535360737)

[20.3 Metadata in <template-body> 74](#_Toc535360738)

[20.3.1 Identifiers 74](#_Toc535360739)

[20.3.2 Generic Metadata: 74](#_Toc535360740)

[20.3.3 Question Metadata: 75](#_Toc535360741)

[20.3.4 QR and LIR Metadata: 75](#_Toc535360742)

[20.3.5 ListItem Metadata 76](#_Toc535360743)

[20.4 Deprecated Metadata: 76](#_Toc535360744)

[21 New and Changed Content in eCC enhanced XML (old) 77](#_Toc535360745)

[21.1 Deprecated XML element 77](#_Toc535360746)

[21.2 New XML attributes 77](#_Toc535360747)

[21.3 Enhanced XML Usage Notes: 78](#_Toc535360748)

[21.4 Comparison of eCC Legacy and enh XML 80](#_Toc535360749)

[22 eCC Implementation Using Composite Keys (@IDs) for eCC Template Items 81](#_Toc535360750)

[22.1 Introduction 81](#_Toc535360751)

[22.2 @IDs and Templates: 81](#_Toc535360752)

[22.3 Data Storage, Terminologies, and Transmission 81](#_Toc535360753)

[22.3.1 Terminology (ICD-O-3 & SNOMED CT) Maps 82](#_Toc535360754)

[22.3.2 Data Storage 82](#_Toc535360755)

[22.3.3 Usage Examples 82](#_Toc535360756)

[22.3.4 Other Uses for @**ID**s 84](#_Toc535360757)

[22.3.5 Data Transmission 84](#_Toc535360758)

[23 eCC Reference Implementation using HTML 85](#_Toc535360759)

[23.1 Automatic DEF Generation 85](#_Toc535360760)

[23.1.1 Table of commonly-used elements and attributes: 86](#_Toc535360761)

[24 Identifier (ID) FAQ 88](#_Toc535360762)

[25 Summary 90](#_Toc535360763)

[26 Glossary 91](#_Toc535360764)

[26.1 Terms and Abbreviations 91](#_Toc535360765)

[26.2 Important SDC Attributes 94](#_Toc535360766)

[27 Data Types 96](#_Toc535360767)

[27.1 Numeric Data Types (reference) 96](#_Toc535360768)

[27.1.1 Restrictions on Numeric Data Types 96](#_Toc535360769)

[27.2 String Data Types (reference) 96](#_Toc535360770)

[27.2.1 Restrictions on String Data Types 97](#_Toc535360771)

[27.3 Date and Time Data Types (reference) 97](#_Toc535360772)

[27.4 Miscellaneous Data Types (reference) 97](#_Toc535360773)

[27.5 Restrictions on Miscellaneous Data Types 98](#_Toc535360774)

# Introduction to Structured Data Capture (SDC)

## Prerequisites (TBD)

## The Lives of Data (TBD)

## What is Structured Data Capture (SDC)?

**Structured Data Capture (SDC)** is a new technology that creates interoperable, computer-readable definitions for standardized **Question/Answer Sets (QAS)** in **Data Entry Forms (DEFs)**.

SDC standardizes the creation and management of DEF QAS items throughout the data lifecycle. The SDC data lifecycle begins with the design of QAS items in DEFs and the inclusion of DEFs in DEF libraries. SDC standardizes the representation of QAS items in a DEF on a computer screen and defines the behavior of DEFs during user-DEF interactions.

After data is captured in an SDC DEF, the SDC model standardizes the transmission and redisplay of captured DEF user responses, as well as re-editing of the original DEF-captured data, and re-transmission of the DEF and its data to other recipients. SDC is also capable of creating specifications for reports based on the captured DEF data, and provides recommendations for the storage and querying of DEF-captured data.

This document describes design of DEFs using the SDC model and touches upon several of the other data lifecycle activities. As a general organizational pattern for this document, we will introduce many topics at a high level early, and then come back to them in progressively more detail.

## SDC’s History and Objectives

The [SDC project](https://oncprojectracking.healthit.gov/wiki/display/TechLabSC/SDC+Home) was initiated by the Office of the National Coordinator for Health Information Technology (ONC) in early 2013 through its Standards and Interoperability (S&I) Framework initiative. Independent SDC-like technologies had emerged previously, but no technology-agnostic standard was available. SDC’s technical workgroups have focused on creating standards by which interoperable forms are defined, rendered, populated and exchanged.

The SDC project was developed in cooperation with [Integrating the Healthcare Enterprise (IHE)](https://en.wikipedia.org/wiki/Integrating_the_Healthcare_Enterprise), a standards organization which emphasizes the interoperability of healthcare information technology (HIT) systems, with a focus on combining constrained standards into profiles for interoperable data transmission. IHE gathers case requirements, identifies available standards, and develops technical guidelines which technical professionals can implement. IHE also hosts yearly “Connectathons” in several countries and stages “interoperability showcases” at [HIMSS](http://www.himss.org/) meetings, at which vendors assemble to demonstrate the interoperability of their products. The SDC workgroup has participated yearly in these IHE activities since 2014.

In keeping with ONC’s role as a standards incubator, in April 2017, ONC transitioned the SDC project to an IHE “community led” project in which many organizations continue to evolve the work incubated by ONC. The version of SDC described in this document supersedes the [October 2016 IHE SDC Profile](http://ihe.net/uploadedFiles/Documents/QRPH/IHE_QRPH_Suppl_SDC.pdf) with improvements made for the Jan 2019 US Connectathon. The SDC development team remains active under IHE and continues to add new and improved features and tools. Over time, new SDC versions will gain powerful new features. This document describes the most important features of SDC form design.

# Overview of SDC Principles

The SDC model is defined by the SDC **XML Schema**, which provides the definitions for creating XML **Form Design File**s (**FDF**s). FDFs are thus **XML instance documents** that conform to the SDC Schema definition. An FDF provides a standardized definition of QAS content and user-interaction behavior for a single DEF, and is designed to be transformed automatically into an SDC-based DEF. The QAS content inside an FDF is intended to support reusable QAS blocks called Data Elements (DEs), which are discussed later. Users’ responses that are captured in the DEF are added to the FDF XML (now called an FDF-Response File [FDF‑R]), and then the responses are transmitted to one or more endpoints.

## SDC Design Principles

A brief review of SDC’s functional and technical requirements will clarify the reason for many design decisions.

* The primary use-cases for SDC are to:
  + Create interoperable clinical data-entry standards for FDFs and DEFs[[1]](#footnote-2)
  + Enable downstream uses (e.g., quality assessments and public health analytics) of the captured standardized data
* FDFs, not terminologies or Common Data Elements (CDEs)[[2]](#footnote-3), are the primary source of context-sensitive semantics.[[3]](#footnote-4)
* SDC uses a single computer-readable information model to standardize DE content in the FDF and DEF. Thus, DEF content is standardized before any user response data is captured by a DEF or data storage device.
* SDC “Form Fillers” (see below) are built to render any SDC-conformant FDF, regardless of the FDF content.
* SDC also supports the definition of report formats, distinct from the DEF layout.
* SDC uses open-source technical standards to define technology-agnostic blueprints for DEF design.
  + SDC has no preferred programming languages.
  + SDC uses industry-standard XML-based mechanisms for the creation and interoperable exchange of FDFs and user responses.
  + SDC uses an interoperable, computer-readable, Schema-defined, XML format to represent the SDC information model, and that allows a computer to build and exchange a wide variety of standardized DEFs.
  + SDC provides an interoperability mechanism for saving and transporting user-entered data inside its original FDF, with 100% round-trip fidelity.[[4]](#footnote-5)
  + SDC includes XHTML support for formatting HTML-based rich text.[[5]](#footnote-6)

## SDC Actors

In the simplest SDC model, there are three primary software **actors** in the SDC ecosystem, each of which is a different kind of node in an SDC transaction network.

1. **Form Managers** (FMs) store FDFs[[6]](#footnote-7) in a repository and transmit them immediately in response to requests from **Form Fillers** (FFs). FMs also have to address issues of authorization and authentication of users, generations of instance IDs, and enforcement of instance IDs and versions (covered later).
2. **Form Fillers** are software applications that:
   1. Retrieve an FDF file from a Form Manager
      1. Alternatively, the FF may retrieve an FDF transformed into HTML, or a URL where the SDC HTML is hosted.
   2. **Render** the XML as a DEF using any convenient programming languages and methodologies. For example, the FDF may be rendered (transformed) to an HTML web page using the XSLT language. Users interact with the DEF inside the FF software.[[7]](#footnote-8)

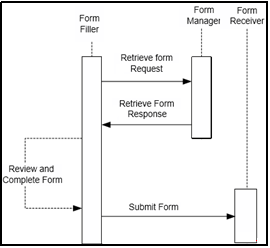


Figure 1

* 1. Capture and validate the user-entered responses in the DEF.
  2. Implement implicit and explicit rules that define the behaviors/actions of the forms in response to user interaction.
  3. Store and/or transmit the captured response data contained inside the original FDF. As noted earlier, an FDF containing user responses is called an **FDF-R**. SDC response data is transmitted as an FDF-R to one or more actors called **Form Receivers** (FRs).[[8]](#footnote-9)

1. **Form Receivers** receive the SDC response data (in an FDF-R file) from the **Form Filler** and process the data according to the FR’s needs. FRs are responsible for storing the captured SDC data as native SDC XML and/or “shredded” into some other storage format. FRs may also be responsible for validation of forms via the SDC Schema, SDC rules, external rules encoded in formats such as Schematrons. Version control, patient matching, authentication and authorization are other potential requirements for FRs.

The above figure shows SDC actors in a sequence diagram. Much more information about FFs, FMs, and FRs may be found in the [IHE Profile document](http://ihe.net/uploadedFiles/Documents/QRPH/IHE_QRPH_Suppl_SDC.pdf), and in the supporting IHE technical files referenced therein. This document primarily describes and discusses SDC Form Filler functions.

**Form Creators** are the most important component of the SDC ecosystem, despite not being IHE actors(transaction nodes). Form creators are individuals and organizations, such as the College of American Pathologists (CAP), The National Cancer Institute (NCI), and Cancer Care Ontario (CCO), that define the content, structure and behavior of FDFs. Form Creators may develop FDFs by working directly with FDF XML, or by using a tool to create/edit FDF files. Form Creators send (e.g., email or upload) FDFs to one or more FMs.

## The SDC Information Model: Data Entry Forms and Data Elements

DEFs can be found in web pages, desktop business software, Electronic Health Record (EHR) forms, or any data-entry screen in any software application. **Data items** live inside DEFs. A data item is a loose concept that encapsulates a question that needs an answer. In the language of SNOMED CT, a data item could be considered similar to a relatively weakly-defined[[9]](#footnote-10) “observable entity.” SDC takes this notion several steps further. SDC standardizes the definition of data items in a DEF by defining the list and/or types of acceptable responses, and organizing multiple data items through the use of an FDF.

An FDF is an XML description of the data items in a DEF. It is not dependent on the programming language used to create a DEF, or the layout/design of the DEF. In other words, the FDF is technology-agnostic and primarily addresses the QAS information content and QAS-interaction behavior of a DEF.

The FDF is a highly-structured arrangement of data items, using standardized XML structures that can be read by a computer to create visible forms on a computer screen. The main purpose of an FDF is to act as a computer-readable *blueprint* for automatically creating the data items inside a DEF. A second function of the FDF is to act as a **transport format** (or “payload”) for the user’s **responses** (answers) entered in the DEF.[[10]](#footnote-11) Thus FDF XML is reused to transmit the FDF-R from the F to the FR. Another function for the FDF is defining the essential metadata needed to create a formatted **report** that will be generated from the DEF responses.

The FDF transmits the user’s responses by adding the responses to the FDF XML and then sending the **FDF-R** (an FDF containing user **R**esponses) to one or more receiving endpoints (Form Receivers). Thus, SDC enables contextual, semantic and syntactic interoperability by exchanging captured user-response data inside the FDF-R.[[11]](#footnote-12) Recipients of the FDF-R can view it as a rendered DEF, view it as a customized SDC report, or parse the data into a secondary format for some other purpose such as data analytics.

### Data Elements

A more formal description of a data item is called a **Data Element** (DE[[12]](#footnote-13)). In the SDC world, DEs are used to define the computer-readable parameters for capturing user-entered data in a DEF. A DE is essentially a question, attached to a definition of acceptable answer choices (either a direct “fill-in” **response** or a list of possible answer choices, called **ListItems**). DEs are a fundamental concept in the SDC model. DEs can live in multiple information-management environments, including, e.g., inside a text document, XML, public DE registry, FDF, DEF, or database. In some cases, terminology codes and many other types of metadata may be added or mapped to DEs. When multiple DE units are nested together into an interdependent unit, the structure is known as a **complex DE**. Many FDFs contain complex DE blocks. SDC also provides an XML standard for representing and transmitting complex DE structure and content.

Since DEs can live in multiple environments with different technical representations (e.g., XML, DEFs and database models), using the term “DE” can be confusing without knowing the context. When found inside a DEF, we will use the term “**Question/Answer Set**” (QAS) for a DE. When found inside an FDF, a DE will be called an **FDF-DE**. When found inside a DE repository/registry, a DE will be called a **Common Data Element** (CDE). A **CDE registry** (or repository or library) allows the sharing of DEs among many forms and allows DE variations and version history to be maintained in a central location. When used by itself, “DE” will refer to all these environments.

### Mapping Data Elements to SDC forms



SDC-embedded FDF-DE units can map to externally-hosted CDEs. As noted above, CDEs are maintained in a CDE registry (repository), independent of FDFs. External CDEs may contain links to a variety of terminologies. This mapping could help make the DEF-captured data easier to communicate, aggregate, compare, retrieve and analyze. CDEs can be used, in principle, to construct SDC forms that share content with other SDC forms. In fact, this is the primary reason that the SDC approach is closely linked to the DE concept. While tools for CDE-based FDF/DEF design are not yet available, SDC has been architected to support this future model from first principles.

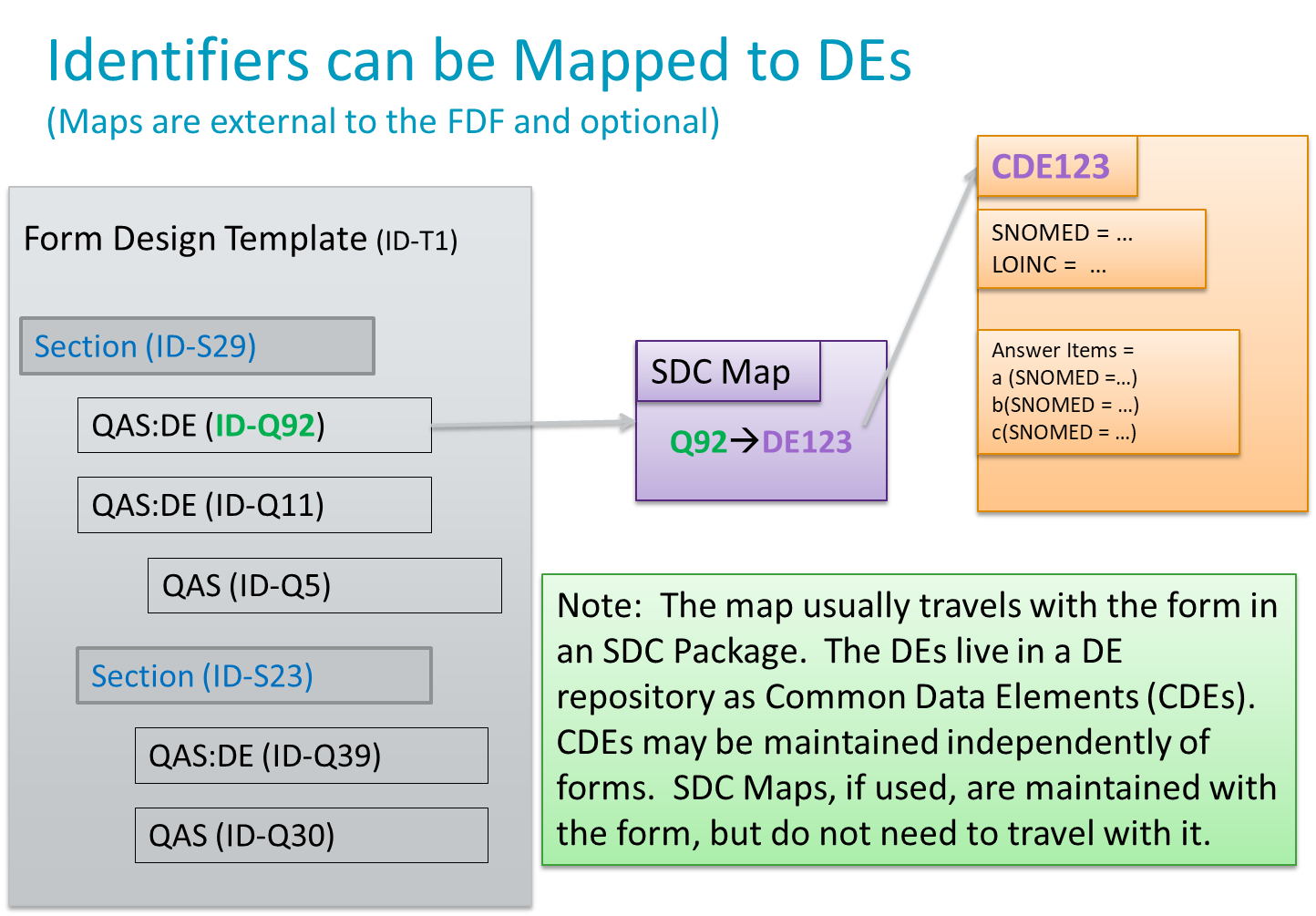
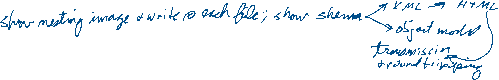


Figure 2: Mapping FDF-DEs to CDEs

Mapping of FDF-DEs to external CDEs may use a DE map inside the FDF XML (using the **CodedValue** element) or may use an external SDC mapping file as a bridge between the FDF and the CDE registry. The latter approach (using a separate mapping file) is generally preferred in SDC, since it makes FDFs and CDE mappings easier to maintain over time. Mapping is accomplished by using a unique identifier from each QAS in the FDF (QAS:DE, a.k.a. FDF-DE) and cross-referencing the unique identifier of the CDE (Figure 2) inside a separate map file.

Codes from terminologies and other coding schemes may be mapped to FDF-DEs in the same ways. However, the SDC model, which is based on DE concepts, prefers mapping directly to DEs rather than to terminology codes.



A further discussion of DEs, CDE registries and terminologies is beyond the present scope of this document.

### The SDC Schema

The SDC Schema expresses an ***information model****[[13]](#footnote-14)* that defines the permitted FDF XML structures. Rather than have one XML Schema for each FDF, SDC uses a single XML Schema that can be used to define an infinite number of different FDFs. The SDC Schema defines the FDF structure using nested DE units, and also provides a rich array of ancillary metadata templates suitable for defining any type of DEF QAS content and behavior. The [SDC Schema files](https://github.com/IHE-SDC-WG/SDC-Schema-Packages) are heavily-annotated and are supported by extensive HTML-based documentation.

# C:\Users\rmoldwi\AppData\Local\Temp\SNAGHTMLf4055a4.PNGData Entry Forms

## Introduction to Data Entry Forms

Each FDF and DEF contains six main types of components: **Question**[[14]](#footnote-15) **(Q)**, **ListItem (LI)** (i.e., answer choices[[15]](#footnote-16)), **Section (S)**, **Displayed Item (DI)** (e.g., notes and other screen text), **ButtonAction (BA)**, and **InjectForm (IF).** The XML FDF components have visible counterparts in the DEF. Q and LI components implement DEs in the DEF. The other components serve mainly to support the DEs, DEF organization and DEF behavior. Each component can store additional hidden metadata which help describe the layout and behavior of the DEF and the resultant report produced from the DEF.

Figure 3 shows a small section from a single FDF, displayed using an HTML DEF. The figure demonstrates S, Q and LI components only. BA and IF items are not shown. Locate each of the following items in the figure:

Change to "untitled"



A thick **dark blue bar** is used for Sections, and **light blue bars** are used for Questions. ListItems (answer lists, displayed with radio/option buttons or check boxes) appear indented beneath the Question bars. Observe that all LIs are children of a parent Question. The **red boxes** in the figure mark the *List* area for each Question, and these boxes contain multiple **LIs**. Some of the LIs are marked as a **ListItemResponse** (**LIR**) component, which is an **LI** with a special *Response* area (an empty rectangular box in the figure) that can receive user-entered text after the **LIR** is selected.

**S1** and **S2** are Sections that contains other types of components. **S1** contains 2 Questions, **QS** and **QR**, and also contains Section **S2**. **S2** contains the Question labeled **QM1**. Containment is indicated by the slight indentation of the child items under the parent item.

Figure 3: Sample DEF

**QR** is a Question that receives a fill-in or **response** value. **QR** is displayed with a white rectangular box representing the user-response area. **QM1** and **QM2** are multi-select Questions, and **QS** is a single-select Question. The last Question, **QM2**, is nested under a **LI** of its parent Question. The nested **QM2** has no title text in the thin, **light blue title bar area**, making it an “untitled Question” (sometimes also called an “invisible Question”). The **LIs** for a multi-select Question are represented here as checkboxes, and single-select LIs are represented as radio/option buttons (circles).

The component nesting structure is critical as it controls the activation of QAS items in the DEF. For example, the untitled question **QM2** and the **QM2** **LIs** are only *activated* when the parent LI is selected (arrow). Conversely, if the **QM2** **LIs** are activated, then we have an implied rule that the parent **LI** must be selected. The term *activated* means that the DEF user can respond to the on-screen Question **QM2** by selecting one or more of the **QM2** **LIs** as an answer choice. The nesting structure therefore implies some significant rule-based DEF activation *behavior*, and as we shall see later, this behavior precisely follows the layout and metadata of the FDF XML.

This was a first introduction to SDC jargon for basic DEF features. We will return to all of these observations in more detail, later in this document.

## Reporting from DEFs

Before diving deeper into FDF details, we need to briefly introduce the concept of **report generation** from DEFs. FDF-based reports can be tailored for target audiences, e.g., for patients versus physician specialists, or tailored to produce short summaries or aggregates of data from multiple sources.

The DEF wording and layout can be used as a guide to design reports; however, this can result in suboptimal report readability. An important issue in report output is that optimal text and layout of Questions and ListItems on the report is not always the same as optimal text and layout in the DEF. The *text* preferred for reports is represented in the SDC metadata using a special Property element feature called **reportText**, to be described later. The *layout* difference between the report and the DEF is highly dependent upon local standards and personal preference, making it difficult to completely specify an optimal layout in an FDF. However, some aspects of the preferred report layout, such as the nesting of data items and display of numeric units, can also be specified using SDC metadata in the FDF. In some cases, FDF metadata are critical to ensure proper rendering and interpretation of reports. We will return to some of these topics at various points later in this guide.

# The Form Design File (FDF)

## Some Important Additional Definitions for SDC FDFs

**XML Form Component (XFC)**: The parts of FDF XML that are used to represent primary DEF controls (see “Control” below) are called *XML Form Components.* Similar to the prior DEF example, XFCs may be one of several types: Question **(**Q**)**, ListItem **(**LI**)**, Section **(**S**)**, DisplayedItem **(**DI**)**, InjectForm **(**IF**)**, and ButtonAction **(**BA**)**. This document will focus primarily on the first 4 XFCs listed above.

Each XFC has a unique identifier attribute (ID)[[16]](#footnote-17). An XFC may also have several other XML attributes and sub-elements specific to a particular XFC type. The ID attribute allows the unique identification of XFCs inside a DEF and also identifies DEF user responses when they are stored as captured data in a database. Every FDF-DE is composed of one or more XFCs. XFCs exist only in FDF XML, but they define the visible items (*controls* or *widgets* – see below) that appear in a rendered DEF.

**Question/Answer Set (QAS):** As noted above, a QAS is a Question in a DEF that includes a definition of its permitted answer(s). QAS items in an SDC DEF are created from XFC “blueprints” present in the FDF. The permitted QAS answers may be *captured* as a *fill-in response* by the user, or by *selecting* from a list of answer choices (ListItem elements). A QAS can be thought of as a DE that lives in a DEF rather than in an FDF or CDE registry.

**Control**: XFCs can be rendered in a DEF in different ways depending on a Form Filler’s programming technique and visual themes/preferences, but all SDC forms use the same SDC information model and XFC-derived form components. When XFCs are rendered in a DEF, the DEF screen objects are called *controls* or *widgets* to distinguish these rendered screen items from the XFCs described by the FDF XML. The selection of appropriate controls for each XFC type is part of the art of programming, and FF designers are free to innovate in this area.

**Metadata:** In this document, the word “metadata” refers to the values of all FDF attributes which affect the rendering, behavior, storage, exchange, reporting and interpretation of FDF form components.

**Data**: In general, we will use the word “**data**” (as distinct from metadata) to refer to the user-entered responses captured in the SDC DEF. User-entered data are also described as “*captured”* user responses. Captured responses (data) arise in two main ways: they may be *selected* from a list of ListItems, and/or entered directly as a response. A directly entered “fill-in” response has an associated data type (e.g., string, integer, binary, etc.) and additional attributes that restrict the valid entries. The FF should validate fill-in (response) data before they are submitted to a FR. Some data may be set as **default** values and may also be marked as **read-only** (or “locked”) in the FDF.

**Context** refers to the hierarchical nesting relationship between form components defined by the FDF XML, and the effect of this relationship on the meaning and interpretation of the nested DE components . Context (especially ancestors) can greatly affect the semantics of the DE units in an FDF and DEF. Incorrectly separating the DEF user responses into separate data slots (e.g., “shredding” captured QAS responses into database tables and fields not designed for SDC) can result in loss of data integrity and could lead to erroneous conclusions. Care must be taken to assure the preservation of the original context. Context is especially important to consider when storing DEF responses and assigning terminologies/codes. Preservation of context is a major advantage of SDC over other technologies. Terminologies (e.g., SNOMED CT codes) are not a reliable substitute for SDC context preservation.

## FDF Structural Overview

### SDC Conventions

We assume the reader has basic familiarity with XML. The reader should be familiar with a few specific SDC XML conventions to understand this document:

**Comments**: To display XML comments inside attribute lists as well as between elements, we use this style.

**Attribute-centric content:** SDC XML elements do not hold text content directly (i.e., between the opening and closing XML element tags). [[17]](#footnote-18),[[18]](#footnote-19) All SDC XML *element content* is contained in *attribute values*, between attribute quotes (see val and myAttribute below), and in *sub-elements* (see **SubElementTag** below). The *definitive* element content (which ordinarily would be text between the opening and closing element tags) is instead placed in a val (value) attribute when appropriate. This XML formatting restriction allows for simpler and leaner XML structures, supports cleaner extensibility using sub-elements, and simplifies the automated generation of programming code to create an SDC object model.

**<ElementTag** val="ElementTag content goes here"**>**

SDC does not allow element content here, between the opening and closing element tags  
 **<SubElementTag** myAttribute="SubElementTag content goes here"**/>**

SDC does not allow element content here, between the opening and closing element tags

**</ElementTag>**  
  
**<ElementTag** val="ElementTag content goes here"**/>**use of @val allows more concise XML

Example 1: Attribute-centric XML content

**XPath notation**: This document will occasionally use XPath notation to concisely indicate nested elements and attributes. For example, **Question\ListField\List\ListItem**\@ID refers to the value of the ID=**"LI1a"** attribute at the end of the highlighted XML path.[[19]](#footnote-20) (In the example, non-essential attributes are omitted for clarity.)

<Question>  
 <ListField>   
 <List>  
 <ListItem ID=**"LI1a"**/>  
 <ListItem />  
 </List>  
 </ListField>  
</Question>

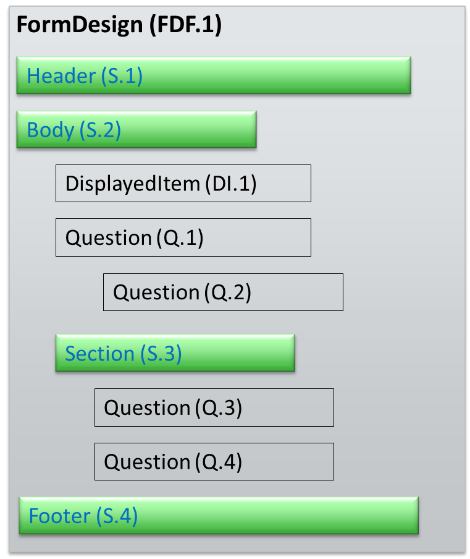
Example 2: XML to demonstrate Xpath notation

**Schema Types:** All SDC elements are modeled with an XML Schema complex Type. For example, a Question element is based on an XML Schema Type called QuestionType. Elements may be discussed by reference to the element itself or its Schema Type. All SDC Schema Type names end with a “Type” suffix to help distinguish the Schema Type from the Element that derives from it. The words “Schema” and “Type” are capitalized to indicate that they represent official W3C terms for XML Schema technology.

**SDC Datatypes**: Almost all SDC datatypes are modeled using the XML Schema datatype model. Thus, every relevant XML Schema datatype is recreated using a new SDC Schema Type modeled after its XML Schema datatype definition. For example, the W3C integer datatype is recreated in the SDC Schema with an SDC datatype called *integer\_DEtype*. The suffix DEtype is a convention to indicate the this SDC Type is used for **D**ata **E**ntry (i.e. to capture user response data) by the DEF user. Each SDC datatype also has a “**S**imple” type, such as *integer\_Stype*. The difference between DEtype and Stype datatype attributes is the inclusion of data entry restriction attributes (e.g. maxExclusive, maxInclusive, totalDigits) in the DE type to constrain what values the user may enter in a DEF. Stype datatypes do not have the DE restrictions. Stype datatypes are used occasionally by form designers to hard-code data into forms in parts of the SDC XML that are not subject to data entry by users and thus they have no need for data entry restrictions. For example, a form designer may add a coded integer value (e.g., 10) inside the SDC XML to annotate a clockface position of selectable ListItem that corresponds to the 10 o’clock position. This coded value would be entered by the form designer using an XML element defined by integer\_Stype.

SDC has 2 datatypes not defined by XML Schema: XML and HTML, both of which exist as DEtype and Stype. These datatypes allow users and form designers to enter custom XML XHTML at various places in SDC forms. Both datatypes require the inclusion of the appropriate namespace in the SDC XML. To validate XHTML in an FDF, the xhtml.xsd Schema must be included, and to validate custom XML, a custom Schema must be provided for that non-SDC XML.

### A First Look at FDF XML:

FDF documents adhere to a basic layout, as shown in Figure 4.

FormDesign and XFC element blocks contains a unique ID in parentheses. The top FormDesign (FDF.1) wrapper element subsumes the top-level sections called Header, Body and Footer. Each section is depicted as a raised green rectangle, to emphasize the role of sections in organizing the FDF and DEF. Inside Body are a DisplayedItem (acting as text on a DEF), a Question (Q.1) that subsumes a second Question (Q.2), and a Section (S.3) that subsumes 2 other Questions (Q.3 and Q.4). The detailed structure of the elements is not shown, but the hierarchical nature of the form layout should be clear.

Please refer to the [SDCFormDesign Schema and sample SDC XML instance documents](https://github.com/IHE-SDC-WG)when studying this section.Concentrate on the Schema parts described in this document, since these have been tested in Connectathons, whereas the other Schema parts may still be under development.

We will now look at a simplified FDF example to study some basic features of the FDF XML layout.

Figure 4: Basic FDF Layout

**<FormDesign**

**Namespace attributes go here:**  
 xmlns="urn:ihe:qrph:sdc:2016"   
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"   
 xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"

**Optional Location of the SDCFormDesign Schema, used for FDF validation:**  
 xsi:schemaLocation="urn:ihe:qrph:sdc:2016 file:SDCFormDesign.xsd"

**SDC FormDesign attributes go here:**  
 order="0"   
 baseURI="sdc.org"  
 lineage="Samples.Samp1"  
 version="v001"  
 ID="Samples.Samp1\_v001\_sdcFDF"  
 fullURI="\_baseURI=sdc.org&amp;\_lineage=Samples.Samp1&amp;\_version=v001&amp;\_docType=sdcFDF"   
 filename="Samples.Samp1\_v001\_sdcFDF.xml"   
 formTitle="Sample Blocks"

**Custom FormDesign Pr**operties go h**ere:**  
 **<Property** propName="ShortName" val="my.SDC.Form"**/>**  
 **<Property** propName="ApprovalStatus val="CTP1""

Main FormDesign content goes here:  
 **<Header** ID="H1" title="This is a Header section (optional)"**/>**   
 **<Body** ID="B1 title="This is the main Body section (required)"**/>**  
 **<Footer** ID="F1" title="This is a Footer section (optional)"**/>**

**<Rules/>**  
**</FormDesign>**

Example 3: Top-level FDF structure

Observe the following basic points about the SDC XML in Example 3:

The top (root) node of an FDF is FormDesign. Listed under FormDesign are many XML attributes, which provide important metadata to support FormDesign.

The attributes may occur in any order. A subset of the possible attributes is shown in the Example. Optional attributess only appear if they contain a value. If an attribute holds its default value as defined in the SDC Schema, then it may not appear in the XML, and is usually omitted to conserve space.

* + The FormDesign attributes are divided into 3 types:
    - Namespace-related attributes, beginning with “xmlns.”
    - An optional schemaLocation attribute that describes the location of the SDCFormDesign Schema, which is useful for FDF validation.
    - Regular SDC attributes: we will cover these later in detail
  + FormDesign is not an XFC, but like an XFC, it does contain a required ID attribute and several other ID-related attributes specific to the FDF root. These will be described later in detail.
  + In addition, FormDesign may contain any number of standard or custom Property elements (described later) to provide essential metadata about the form.
* FormDesign wraps three important nodes: Header, Body and Footer. These nodes may appear only once in the FDF, always under FormDesign. These three nodes are just top-level Section XFC nodes that have special descriptive names because have specialized functions.
  + When the FDF is rendered as a DEF, the Header section “sticks” to the top of the DEF,[[20]](#footnote-21) the Footer section sticks to the bottom of the DEF, and the Body section is the main middle section of the DEF. The Body section may be scrolled by the user, but the Header and Footer must always appear on the DEF screen.
  + The Body section is *required* in the FDF, but Header and Footer are optional. The Header, Body and Footer may contain nested XFCs (including other sections), and this nesting may continue to any depth.
  + The optional title attribute on each element contains the text that will appear in the DEF.
  + Since each of these three nodes is a kind of Section XFC, they each require a unique ID value.
* The Rules element at the bottom holds instructions that control some of the behavior of an FDF as the user interacts with XFCs in the form.
  + Many FDF behavioral instructions can also be attached directly to the XFCs, and do not appear under Rules. The instructions that appear here are mainly those that affect interactions between multiple XFCs that can located in widely-separated areas of the FDF’s XML tree.

Each of the SDC elements and attributes will be explained later in more detail.[[21]](#footnote-22)

### The XFCs

We will briefly survey the XFCs in this section in order to obtain a rapid overview, but we will return to them in more detail later.

**<Section** ID="S1" title="I am a section" **>**  
 **<ChildItems>**  
 **<DisplayedItem** ID="DI1" title="I am a note on the screen"**/>**  
  
 **<Question** ID="Q1" title="This is a question title"**>**  
 **<ListField>**  
 **<List>**  
 **<ListItem** ID="LI1" title="ListItem.title.1"**/>**  
 **<ListItem** ID="LI2" title="ListItem.title.2"**/>**  
 **<ListItem** ID="LI3" title="ListItem.title.3"**/>**  
 **</List>**  
 **</ListField>**  
 **</Question>**  
 **</ChildItems>**  
**</Section>**

Example 4: XFC Nesting with a ChildItems Element: Section, DisplayedItem, Question and ListItem

Example 4: XFC Nesting with a ChildItems Element: Section, DisplayedItem, Question and ListItem shows a simple SDC XML snippet representing the 4 most common XFCs: Section, DisplayedItem, Question and ListItem, along with a minimum number of SDC attributes. This is one of the simplest SDC DE structures in an FDF: a Section that wraps a single DisplayedItem and a Question. For now, note that the following features:

* A DisplayedItem and a Question are subsumed by a ChildItems element.
* The Question structure subsumes a ListField and ListItem element before the ListItem elements appear.
* Each XFC has an ID and a title attribute.

#### XFC Nesting in SDC

ChildItems: The ChildItems element in Example 4: XFC Nesting with a ChildItems Element: Section, DisplayedItem, Question and ListItem provides a hierarchical nesting wrapper for S, Q and LI; these XFCs can become parent XFCs. A parent XFC must use a ChildItems element to wrap child XFCs. The ChildItems wrapper element serves to separate XFC descendants from other parent-owned metadata elements (such as Property, Comment and Event elements) that supplement the parent XFC. S, Q and LI may be nested to any depth.

Question does not use the ChildItems wrapper for containing ListItems. Instead, ListItems appear in a \Question\ListField\List\ListItem XML construct without a ChildItems wrapper. This exception exists because the ListField\List structure serves as a wrapper for all ListItems belonging to a Question element.

Any XFC, except for LI, may be nested under a ChildItems element. Thus, a child Section or Question or ListItem can also wrap one or more XFCs if they are contained in a ChildItems element.

The general rules for XFC nesting are:

* The following XFCs may have descendant XFCs S, Q and LI. These XFC descendants are always wrapped inside a ChildItems parent wrapper.
  + DI, BA, IF do not support descendant XFCs. However, IF may inject SDC XML that contain descendants (described later).
* The following XFCs must have a direct ChildItems parent: S, Q, DI, BA, IF.
  + The only exceptions are the Header, Body and Footer sections that exist under the FormDesign element.
* LI is an exception to the other XFCs. An LI always appears in a \Question\ListField\List\ListItem construct

#### The XFC Building Blocks

Please note the following basic XFC concepts and we will cover them in detail with more examples.

* Only Question and ListItem are involved directly in DEF data capture.
* Section and DisplayedItem are used to support display, styling, and organization of the DEF.
* Only Section, Question and ListItem can subsume nested child XFCs using the ChildItems element.
* ListItem has a close structural relationship with Question, which will be described below.
* SDC supports user-controlled repeating of Section and Question XFCs, which allow a user to enter data for the repeating Questions multiple times, when appropriate. Any SDC XML that is subsumed by the repeating XFC elements is repeated with them.

We will now review just enough information about each XFC element to get a broad understanding of the basic FDF structure. We will return to each XFC in detail later.

Section **(**S**)**: The most basic structure of an FDF is the Section XFC. As noted above, three specially-named sections are the Header, Body and Footer, which appear as direct children of the top-level FormDesign element.[[22]](#footnote-23) No other XFCs are allowed directly under the FormDesign root element. The main purpose of the Section XFC is to group other XFCs (including other Sections) into blocks of content that make sense for users of a DEF. Section, along with any subsumed XML, optionally may be repeated inside the DEF.

DisplayedItem **(**DI**)**:

The DIelement is used to define visible text[[23]](#footnote-24) for display almost anywhere in a DEF. Each DI has a unique ID, but, unlike Section and Question, a DI cannot have XFC descendants, and cannot be repeated subsumed inside a repeating XFC (S, Q, or LI). Displayed text that requires the use of XFC descendants must use the Section element instead. The formatting of any SDC element, such as DI or Section, can be specified by the styleClass[[24]](#footnote-25)attribute.

Question **(**Q**)**:

Questions may appear in two basic forms.

* Questions that can capture a response (fill-in) value directly are called Question**-**Response **(QR) Items.** They are sometimes also called Question Fill-in (QF) items. Response metadata and captured response values are handled by the ResponseField\Response XML element structure.
* Other Questions take a list of answer choices, called ListItems (LI). In the most common cases, Questions with ListItem choices are either **single-select** (**QS**) or **multi-select** (**QM**). QS and QM items can never have a ResponseField\Response element under the Question.

ListItem **(**LI**):**

ListItems come in two basic types:[[25]](#footnote-26)

* The simple ListItem is sometimes called an “answer choice” or “pick list item.”
* A ListItem that, when selected, can capture a response, is called a ListItemResponse item (LIR), but may also be referred to as an “answer fill-in” (AF). An LIR is a LI that subsumes a ListItemResponse element structure in the FDF XML. The ListItem Response structure is ListItemResponseField\Response. We will cover the Response element later.
* ListItems are always wrapped together as a group, nested inside a Question\ListField\List element structure.[[26]](#footnote-27)

ButtonAction **(**BA**):**

TBD

All of the XFCs can trigger actions in the DEF that affect the appearance, behavior and data responses in the DEF. However, ButtonAction provides a user-controlled visible screen area (a “button” control) to trigger any kind of action in a DEF.

InjectForm **(**IF**):**

TBD

InjectForm is a kind of “virtual XFC” because it acts as a placeholder where XFCs from the same FDF or other FDFs may be injected into any location in the FDF. Even an entire FDF can be injected inside the InjectForm element.

## FormDesign Attributes and Properties

The FDF root element (FormDesign or DemogFormDesign) has a relatively large number of attributes. Some of the attributes address ***static*** FDF metadata inserted by the FDF designer. These are related to the identity of the FDF, such as the ID or the filename.

Other attributes have values that are used to track form ***instance*** data used for tracking individual instances of forms that are instantiated for a specific purpose, e.g., entering data on a specific patient, during a specific encounter. Instance values are assigned by the FM, FF, and/or FR.

For tracking purposes in some use cases, a FM may assign some of instance identifiers before delivering an SDC form to the FF, and the FF must check for these pre-assigned instance-tracking values to avoid inappropriately overriding FM-assigned values. In other cases, the instance tracking values are assigned by the FF, and included as part of the forms data set that is transmitted to the FR. If the FF detects that the FM has not assigned instance values to the instance attributes, then it may assign them itself. A FR should not alter the contents of a received FF form. However, a FR may log some or all of the instance attribute values in another data storage area, and may add its own logging information (e.g., time of receipt, hash value of XML contents, etc.), but these data should not be added to the FDF-R. In many cases, the FF is a “dumb” web page, and therefore performs none of the instance functions. In some cases, the FM and/or FF and/or FR may be part of the same institution, and this will help determine which actor(s) assign the instance values.

All instance attributes are optional in the XML, because they are not relevant in empty (not-instance) forms. Some static attributes are also optional. A FR should check for the correct assignment of all FormDesign attribute values as part of the validation process of a received form.

A third class of attributes are the ***instance status*** attributes. These are assigned by the FF based on information provided by the DEF user. The attributes address the type of changes in the DEF and finality of the submitted FDF-R.

All required attributes are marked with required.

**<FormDesign**

Namespaces  
 xmlns="urn:ihe:qrph:sdc:2016" required   
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"   
 xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"  
 xmlns:h=<http://www.w3.org/1999/xhtml>  
  
Schema-based validation   
 xsi:schemaLocation="urn:ihe:qrph:sdc:2016 file:SDCFormDesign.xsd"

generic BaseType attributes  
 order="0"

type=""

styleClass=""

name="Lung.Bmk.227"

identifier-related attributes

baseURI="cap.org" required

lineage="Lung.Bmk.227" required

version="1.001.011.RC1" required

ID="Lung.Bmk.227\_1.001.011.RC1\_sdcFDF" required  
 fullURI=  
 "\_baseURI=cap.org&amp;\_lineage=Lung.Bmk.227&amp;\_version=1.001.011.RC1&amp;\_docType=sdcFDF" required

prevVersionID=

"\_baseURI=cap.org&amp;\_lineage=Lung.Bmk.227&amp;\_version=1.000.001.REL&amp;\_docType=sdcFDF"

miscellaneous helper attributes  
 filename="Lung.Bmk.227\_1.001.011.RC1\_sdcFDF.xml"

basedOnURI=""

formTitle="Lung Biomarker Reporting Template"

data-submission identifiers  
formInstanceURI=""

formInstanceVersionURI=""

formPreviousInstanceVersionURI=""

FormDesign Property elements  
 **<Property** name="Copyright" type="CAPeCC\_static\_text" styleClass="copyright" order="1"  
 propName="Copyright"  
 val="(c) 2018 College of American Pathologists. License required for use."**/>**  
 **<Property** name="ApprovalStatus" type="CAPeCC\_meta" order="13" propName="ApprovalStatus"  
 val="RC1"**/>**

Main section of form starts here  
 **<Body** name="Body" order="14" ID="Lung.Bmk.227\_1.001.011.RC1\_sdcFDF\_Body"**/>**   
**</FormDesign>**

Example 5: FormDesign attributes and Properties

### FDF Namespaces

Handled later

### FormDesign Attributes

Static Attributes: Values for these attributes are supplied by the form designer.

* formTitle**:** Human readable title for display when choosing forms from list provided by a FM. The formTitle may be displayed at the top of the DEF
* baseURI: This parameter is required in the FormDesign element but is optional in the XFCs. It identifies the organization that is responsible for designing and maintaining the FDF or XFC. It’s best to avoid using prefixes like “**Error! Hyperlink reference not valid.**” or “**Error! Hyperlink reference not valid.**” because these can occasionally cause XML validation errors when used in a URI-typed field.
* basedOnURI**:** URI used to identify the SDC form that that the current FDF is based upon. In most cases, this should be a standard SDC form that is modified and/or extended by the current FDF. It’s best to avoid using prefixes like “**Error! Hyperlink reference not valid.**” or “**Error! Hyperlink reference not valid.**” because these can occasionally cause XML validation errors when used in a URI-typed field.
* lineage: A string identifier that is used to group multiple versions of a single form. The lineage is constant for all versions of a single kind of form. When appended to baseURI, it can be used to retrieve all versions of one particular form. Example: lineage="Lung.Bmk.227"
* version: A string that contains the version text for the current form. It is designed to be used in conjunction with baseURI and lineage.
* ID: In FormDesign, the ID is used to uniquely identify an “empty” FDF (i.e., *not* an FDF-R, which contains user-entered data) from other FDF files that have different FDF content. Thus, a FormDesign ID is shared by all FDFs that have the same lineage and version. TO represent this concept, we create the ID with the following formula:   
  ID=lineage\_version\_sdcFDF.   
  In other words, we concatenate the components lineage, version and the text “sdcFDF” using “\_” characters as separators. The suffix “sdcFDF” indicates that we are working with the FDF variant, not the FDF-R or the DE variant of SDC XML. From Example 5: FormDesign attributes and Properties:   
    
  ID="Lung.Bmk.227**\_**1.001.011.RC1**\_**sdcFDF",   
    
  we can clearly see how lineage and version values and the text “sdcFDF” are concatenated with “**\_**”.
* filename**:** The filename of the FDF when is saved to a file storage device (e.g., a disk or USB drive). The filename appears inside the FDF XML to help ensure the identity of the FDF content in case the saved filename (on a disk drive, etc.) has been changed for any reason.
  + For FDF files without responses, the filename, one suggested format is lineage\_version\_sdcFDF.xml.
  + For assigning a filename to FDF-R instance documents, the following filename format may be considered: lineage\_version\_*instance*\_*instanceVersion*\_sdcFDFR.xml. *instance* is a GUID that identified the al versions of an instance FDF-R document. *instVer* is a version of the instance FDF-R. However, this format can become rather long, and thus short GUIDs or other formats may be considered.
* fullURI: The full URI that uniquely identifies the current form. It is created by building up a REST-style query string from several components: baseURI, lineage, version , and **doctype,** using a format of component=value pairs, as in the following example:   
    
  fullURI="**\_baseURI**=cap.org&amp;**\_lineage**=Lung.Bmk.227&amp;**\_version**=1.001.011.RC1&amp;**\_docType**=sdcFDF".   
    
  Note that each bolded component name is preceded by an underscore “**\_”** and most component names and values derive from a FormDesign attribute. The only fullURI component not present as an attribute in FormDesign is **docType**, which is always set to “sdcFDF” for FDFs, and is set to “sdcFDFR” for FDF-R documents, which contain data. Note that all “&” (ampersand) symbols are escaped using the standard XML/HTML “&amp;” escape notation.
* prevVersionURI**:** ThisURI is used to identify the SDC form that is the immediate previous version of the current FDF. The format is the same as fullURI. The primary role of this optional attribute is to allow automated comparisons between a current FDF and the immediate previous FDF version. This is often helpful when deciding whether to adopt a newer version of an FDF.

**Instance Attributes:** These are attributes found in an FDF-R, which is created to contain user-entered data. Included in brackets “[ ]” are the IHE actor(s) that are responsible for adding the correct value to each attribute. Different use cases may redefine the requirements for each attribute.

* formInstanceURI**:** [FM or FF] Unique URI used to identify a unique instance of an FDF that contains user responses. It is used for tracking form responses across time and across multiple episodes of editing by end-users. This instance identifier does not change for each edit session of a form instance. The formInstanceURI is formatted similarly to the fullURI but includes an additional parameter: \_instance= {some GUID}. A GUID (e.g., a1b2c3d4…) should be used to create the unique instance part of the URI. An example is:   
    
  formInstanceURI="**\_baseURI**=cap.org&amp;**\_lineage**=Lung.Bmk.227&amp;**\_version**=1.001.011.RC1&amp;**\_docType**=sdcFDF&amp;**\_instance**=Abc1dee2fg987".   
    
  It is also possible to create the URI from a GUID alone, but that approach is clearly less helpful to the human reader.
* formInstanceVersionURI**:** [FM or FF] Globally-unique URI used to identify a unique edit session of a form's saved responses. It is used for tracking form responses across time and across multiple episodes of editing by end-users. This identifier must change for each edit/save session of a form instance (defined by formInstanceURI).   
  The formInstanceVersionURI should be formatted similarly to the formInstanceVersionURI but must include an instance version identifier (\_*instVer*) in the URI: \_*instVer*={some version}. A suggested approach for the version information is the use of the release date/time for the new version, in W3C datetime format. In general, all transmitted versions should be archived by the FR or other storage system(s). An example is   
    
  formInstanceVersionURI="**\_baseURI**=cap.org&amp;**\_lineage**=Lung.Bmk.227&amp;**\_version**=1.001.011.RC1&amp;**\_docType**=sdcFDF&amp;**\_instance**=Abc1dee2fg987&amp;**\_instVer**=2019-07-16T19:20:30+01:00”.   
    
  It is also possible to create the URI from \_instance and \_instVer parameters alone, as long as the URI is able to uniquely identify and retrieve the instance and version of the globally unique FDF-R that was transmitted.
* formPreviousInstanceVersionURI**:** [FM or FF] Unique URI used to identify the immediate previous instance of a form that contains responses. The formInstanceVersionURI of the previous instance is copied into formPreviousInstanceVersionURI for the current editing session, and the formInstanceVersionURI receives a new version suffix before the form is saved/transmitted. This attribute is used for tracking responses from a single form instance across time and across multiple episodes of editing by end-users. The formInstanceVersionURI value must change for each edit session of a form instance.

**Instance Status Attributes:** The following instance attributes are generally assigned by the FF, based on input from the DEF user:

* approvalStatus: [FF] Describes report fitness for clinical or other action: **inProcess**: currently being edited, users should not rely on results; **preliminary**: report is awaiting final review and approval; **approved**: report is fit for clinical or other action; often synonymous with final; **cancelled**: report/procedure has been aborted before issued; **retracted**: report has been deemed unfit for clinical or other action
* completionStatus: [FF] The extent to which a report contains all of the requested information: **pending**: no information is yet available; **incomplete**: some requested information is not yet available; **complete**: all information is available in the requested report
* newData: [FF] Identifies existence of data that is new to the current instance of package, form, section, or question compared to the previous instance of the package, form, section, or question. Values: **True**/**False**
* changedData: [FF] Identifies existence of data that has been changed in the current instance of package/form/section/question compared to the previous instance of the package/form/section/question. Values: **True**/**False.**

### FDF Identifiers

Each FDF has its own ID attribute in the FormDesign element, which uniquely identifies each “class” of FDF document. The word “class” is meant to convey that the lineage and version are identical for each FDF that shares an ID; The XML content is identical in FDFs that share an ID.

In addition, every XFC in an FDF contains an ID value, which must be unique within its FDF. However, ID values do not need to be **globally** unique. Therefore, the same ID values may be used in other forms, including: Forms with the same lineage and forms form a different source. Within a given organization, it is best to control the release of IDs, so that the same ID never appears in a different FDF lineage.

Most XFC IDs are not changed when an FDF from a given lineage is released with a new version. However, a new FDF version will force an ID change for the FormDesign element. Within a given lineage of FDFs, identical XFC ID values signal that the semantics and basic structure of the object remains unchanged across different versions. The criteria for changing IDs on versioned Q and LI items are discussed later in this document.

It is possible to create a system for globally unique ID values that would enable every FDF and every XFC in a form to be uniquely identified, e.g., by using GUIDs. However, the use and maintenance of long GUID ID values can sometimes cause difficulties and errors.

To enable the use of simpler, shorter ID values that are easier to use, the SDC Schema includes the baseURI attribute. The baseURI functions similarly to an XML namespace since it uniquely identifies the organization or group or even the current form. When the baseURI value is used in conjunction with the ID value, a composite globally-unique identifier (**CGUI**) is created for each XFC. For example, if the simple ID value is "100", and the baseURI is "www.cap.org/FormsDomainSDC1/", then the CGUI is " [www.cap.org/FormsDomainSDC1/100](http://www.cap.org/FormsDomainSDC1/100)", which the issuing organization (cap.org) should guarantee as unique within an FDF lineage. The CGUI is generated as needed and thus is not found explicitly in the FDF XML. Note that the ID value “100” need not be globally unique, but the CGUI should be globally unique.

The default XFC baseURI is defined as identical to the FormDesign baseURI. A new baseURI is not assigned to an XFC unless the FormDesign baseURI is not “appropriate” for a specific XFC and its descendant XFCs. An appropriate baseURI should reflect the organization that created and/or maintains the XFC content, as well as the FDF content domain that it addresses. The baseURI is inherited by all descendant XFCs and should not be added to the SDC XML unless there is a change in baseURI.

Ideally, the baseURI is assigned once at the highest level of an FDF (the FormDesign element). Every descendant ID then inherits the baseURI, without repeating the baseURI at each XFC. If necessary, multiple baseURIs may be used throughout an FDF; these override any higher-level URI assigned above it in the FDF hierarchy. Thus, an XFC uses the baseURI ancestor that is closest to it in the FDF hierarchy, starting with its own baseURI value (if it is present).

To create a baseURI, the institution that creates an SDC form should have one or more registered globally-unique IDs (e.g., domain names or GUIDs) that are used to uniquely[[27]](#footnote-28) identify the origin and uniqueness of its SDC forms. Ideally, these IDs should be in URL format, and should ideally represent real URLs that can be "dereferenced" to provide information about the organization and its forms.[[28]](#footnote-29) It’s best to avoid using prefixes like “http://” or “https://” in the baseURI text because these add unnecessary length and can occasionally cause XML validation errors when used in a URI-typed field such as formInstanceURI.

### FormDesign Properties (eCC)

Earlier, we introduced the concept of Property elements under the FormDesign tag. FDF Properties allow form designers to introduce domain-specific metadata into the FDF for a variety of purposes. Any of these Property elements may be displayed (or not) in the FDF, depending on the use case, and under the control of the FF software.

To provide some examples, we list the CAP eCC Property types for the FormDesign element:

**<Property** name="Copyright" type="CAPeCC\_static\_text" styleClass="copyright"

propName="Copyright" val="(c) 2018 College of American Pathologists. All rights reserved. License required for use." **/>**  
**<Property** name="GenericHeaderText" type="CAPeCC\_static\_text" propName="GenericHeaderText" val="Surgical Pathology Cancer Case Summary (Checklist)" **/>**  
**<Property** name="Category" type="CAPeCC\_meta" propName="Category" val="Endocrine" **/>**  
**<Property** name="OfficialName" type="CAPeCC\_meta" propName="OfficialName" val="ADRENAL GLAND" **/>  
<Property** name="CAP\_ProtocolName" type="CAPeCC\_meta" propName="CAP\_ProtocolName" val="Adrenal Gland" **/>**  
**<Property** name="CAP\_ProtocolShortName" type="CAPeCC\_meta" propName="CAP\_ProtocolShortName" val="Adrenal" **/>**  
**<Property** name="CAP\_ProtocolVersion" type="CAPeCC\_meta" propName="CAP\_ProtocolVersion" val="4.0.1.1" **/>**  
**<Property** name="TemplateID" type="CAPeCC\_meta" propName="TemplateID" val="129.100004300" **/>**  
**<Property** name="Restrictions" type="CAPeCC\_meta" propName="Restrictions" val="Please refer to the cancer protocol cover page (www.cap.org/cancerprotocols) for information about which tumor types and procedures can be reported using this template." **/>**  
**<Property** name="CAP\_Required" type="CAPeCC\_meta" propName="CAP\_Required" val="true" **/>**  
**<Property** name="AccreditationDate" type="CAPeCC\_meta dt.dateTime" propName="AccreditationDate" val="2/28/2018 12:00:00 AM" **/>**  
**<Property** name="WebPostingDate" type="CAPeCC\_meta dt.dateTime" propName="WebPostingDate" val="6/30/2017 12:00:00 AM" **/>**  
**<Property** name="ShortName" type="CAPeCC\_meta" propName="ShortName" val="Adrenal.Bx.Res" **/>**  
**<Property** name="ApprovalStatus" type="CAPeCC\_meta" propName="ReleaseStatus" val="RC2" **/>**  
**<Property** name="AJCC\_Version" type="CAPeCC\_meta" propName="AJCC\_Version" val="8th Edition" **/>**



CAP Property types:

* Copyright: A copyright statement
* GenericHeaderText: Text that appears at the top of the DEF
* Category: The organ group that includes the current form, e.g., “Endocrine”
* OfficialName: The full human-readable name of the current form
* CAP\_ProtocolName: The name of the CAP Cancer Protocol that contains the current form
* CAP\_ProtocolShortName: The abbreviated name of the CAP Cancer Protocol that contains the current form
* CAP\_ProtocolVersion: The version of the CAP Protocol
* TemplateID: A numeric identifier for the form lineage, appended to the lineage text
* Restrictions: Rules about when to use or not use this form
* CAP\_Required: The value is “true” if the form is required for Commission on Cancer accreditation
* AccreditationDate: The data that this form must go into effect to satisfy the requirements of accreditation-related surveys
* WebPostingDate: The date the form was posted on the CAP website
* ShortName: A concise name for the form, more manageable than the long OfficialName Property
* ApprovalStatus: A short text flag that indicates how close the form is to an officially-approved release. Examples include **CTP** (Community Technology Preview) , **RC** (Release Candidate) and **REL** (official Release for implementation)
* AJCC\_Version: The version of the American Joint Committee on Cancer (AJCC) Staging Manual used in the FDF

# Introduction to SDC Basic Schema Types



## SDC Schema File Overview

## Schema files.

The basic SDC Schema that is used for designing FDFs is organized as a hierarchy of 5 files. Each file includes the files that precede it in the hierarchy. The lowest level is the SDCBase.xsd, and the top level is SDCFormDesign.xsd. The hierarchy is arranged as follows, with the most inclusive files on top:

**SDCFormDesign**.xsd - includes:  
 SDCExpressions.xsd - includes:  
 SDCResources.xsd - includes:  
 SDCDataTypes.xsd - includes:  
 SDCBase.xsd

Covered later (TBD) are additional SDC Schemas required for SDC retrieval transactions:

**SDCRetrieveForm**.xsd – includes:  
 SDCFormDesign.xsd  
 SDCTemplateAdmin.xsd  
 SDCMappings.xsd

And one Schema used for FDF and FDF-R transmissions to FRs

**SDCSubmitForm** – includes:  
 SDCFormDesign.xsd

At this point, we will be concerned only with **SDCFormDesign** and its sub-Schemas.

### Basic SDC Schema Type Hierarchy

To best understand the attributes and features supported by the various XFCs, we need to consider the SDC Schema inheritance model. Note that the word “Schema” is capitalized, which, in this document, indicates that we are dealing with an official W3C version 1.0 XML Schema to define the SDC architecture.

We now need to look at the layered Schema Types from which the XFCs derive. In the SDC Schema architecture, all XML elements are backed by an XML Schema Type to better support automated code generation from the Schema. This document will often use the capitalized word “Type” to indicate that we are discussing the element’s definition as Schema Type, and not the element that derives from it and which is found in the SDC XML (i.e., in the FDF). In SDC, all Types and elements (except for W3C-derived datatypes) are capitalized, and all attributes use camelCase.[[29]](#footnote-30)

The hierarchical arrangement of the SDC Schema Types within the Schema files is essential to understanding inherited attributes and elements. It is important to understand that the hierarchical arrangement of Schema files is not the same relationship as the hierarchical arrangement of Types within the and between the Schema files. SDC Schema Types are derived by inheritance from and aggregation of more primitive Types in the various included files in the Schema hierarchy.

Note that *abstract* Types are Schema Types that are never directly used to derive XML elements in any SDC XML file. However, new Schema Types may be derived from an *abstract* Type, and these new Types may be used to define an XML element.

SDC *datatypes* include almost all W3C datatypes, with the addition of special SDC types that subsume HTML and XML blocks inside an FDF. These specialized HTML and XML blocks obey other Schema definitions, and thus must have their own XML namespaces to be included in SDC XML documents. HTML blocks must use the xhtml Schema, which is included with the SDC Schema set.

For consistency with W3C naming conventions, W3C *datatype* element names begin with *lower* case letters, unlike all of the other native SDC elements, which begin with *upper* case letters.



We now introduce the SDC Schema Types. In the following SDC Schema Type hierarchy (Example 6), the most primitive types are on top, and the most derived types are on the bottom. We present the Types in this manner to mirror a common approach to represent object trees in Object-Oriented Programming (OOP) languages, where each child object type points upwards to its “parent” (i.e., more primitive) type. For example, a class diagram[[30]](#footnote-31) for an object-oriented programming diagram will sometimes display the tree in the reverse orientation, with the most derived “top-level” types on top. We follow this convention to highlight the deep connection between the SDC Schema Type definitions and the programming objects that may be created automatically from each Type.

To avoid confusion with the explanations used for Type trees, we need to review some terminology. The Types towards the top of the hierarchy are described with terms such as “parent,” “base,” “primitive,” ”supertype” and “low-level.” Some terms like “low-level” and “base” can be confusing to some people because the low-level (base) types are at the top of the tree. These terms are used because they are the low-level building blocks from which more “higher-level” complex Types are derived and assembled. The Types towards the bottom of the are thus described with terms like “high-level,” “child,” “subtype,” “descendant,” and “derived.” In some depictions of Type hierarchies, the top-bottom orientation of the tree is reversed, with the primitive types at the bottom, and the high-level (derived) types at the top.

|  |
| --- |
| *BaseType (abstract)*  (all SDC Datatypes and derivatives included in the types listed below)  CodeMatchType  CommentType  ExtensionType  *ExtensionBaseType (abstract)*  (all Event, Rule, Action types and subtypes)  (all items defined in the SDCRetrieveForm sub-Schema)  PropertyType  TemplateAdminType and most subtypes  ChildItemsType  LinkType, BlobType, ContactType and most subtypes  CodedValueType, CodeSystemType  ListFieldType, ListType, LookupEndpointType  ResponseFieldType, ListItemResponseFieldType  *IdentifiedExtensionType (abstract)*  DataElementType  FormDesignType  **InjectFormType** (InjectForm)  ***DisplayedType*** (DisplayedItem)  **ButtonItemType** (ButtonAction)  **ListItemType** (ListItem)  *RepeatingType (abstract)*  **SectionItemType** (Section)  **QuestionItemType** (Question) |

Example 6: SDC inheritance model

See the SDC Schema at … and the HTML documentation at …

Four of the Types are marked with “(*abstract)”*, which means that an XML element cannot be created from that Type; an XML element can only be created from a non-abstract “*concrete*” Type that inherits (derives) from the *abstract* Type, further down the hierarchy tree.

The XFC Schema Types,[[31]](#footnote-32) which are used to create SDC XML elements, are shown in **bold**. XFC Type names are followed by parentheses containing the name of the element in the FDF XML, e.g., **ListItemType** (ListItem).

Five of the Types listed above are **parents** (shown in *italics*), i.e., they have 1st level descendants that inherit from them, and which pass their elements and attributes to all of their 2nd level descendants, and so on, through multiple levels of inheritance.

Finally, note that DataElementType and FormDesignType derive directly from *IdentifiedExtensionType (abstract)*. This derivation introduces a level of common ancestry between these two high-level Types and all the XFCs. The common ancestry is critical to the creation of a consistent and coherent SDC object model from the SDC Schema.

## The SDC Schema Inheritance Model and XFC Definitions

The following Example 7 shows a partial inheritance hierarchy “tree” of the main SDC Schema Types, derived from Example 6: SDC inheritance modelExample 6. As before, the Types at the top of the hierarchy are the most primitive and the elements and attributes defined by the Type are inherited by Types further down the hierarchy tree. Indented types inherit elements and attributes from the parent Type above it, and all parallel-aligned Types are siblings that inherit from the same parent Type.[[32]](#footnote-33)

Note that all XFCs inherit from *BaseType* and *IdentifiedExtensionType*. All XFCs, except for **InjectFormType**, inherit from ***DisplayedType***. **SectionItemType** and **QuestionItemType** additionally inherit from *RepeatingType*.

*BaseType (abstract)*

*ExtensionBaseType (abstract)*

*IdentifiedExtensionType (abstract)*

**InjectFormType** (InjectForm)

***DisplayedType*** (DisplayedItem)

**ButtonItemType** (ButtonAction)

**ListItemType** (ListItem)

*RepeatingType (abstract)*

**SectionItemType** (Section)

**QuestionItemType** (Question)

Example 7: SDC Schema XFC Inheritance Hierarchy

The list below again enumerates the lower-level SDC Types (numbered 1-5). Underneath each Type is a bulleted line that lists attributes and elements introduced at each Type. Omitted from the Example are the five XFC Type descendants that have no descendants of their own (i.e., all XFC Types except ***DisplayedType***). All of the attributes and elements in the upper parts of the Type hierarchy structure are inherited by the derived types further down the list. Of all the inherited elements and attributes listed below in Example 8, only ID is required to be present in the FDF for every XFC in the FDF XML.

|  |
| --- |
| 1. *BaseType (abstract):*  * name, styleClass, type, order  1. *ExtensionBaseType (abstract):*  * Comment, Extension, Property  1. *IdentifiedExtensionType (abstract):*  * ID, baseURI  1. ***DisplayedType (****DisplayedItem****):***  * Link, BlobContent, Contact, CodedValue, OnEnter, OnExit, OnEvent, ActivateIf, DeActivateIf * title, enabled, visible, mustImplement, showInReport  1. *RepeatingType (abstract):*  * minCard, maxCard, repeat, instanceGUID, parentGUID |

Example 8: Inherited XFC Elements and Attributes

To obtain the full list of inherited elements and attributes for the *RepeatingType*, we walk up the inheritance hierarchy to identify the ancestral elements and attributes. Example 9 shows the elements and attributes inherited by Section**[[33]](#footnote-34)** and Question elements, which are derived from *RepeatingType* (as shown previously in Example 7):

* name, styleClass, type, order
* Comment, Extension, Property
* ID, baseURI
* Link, BlobContent, Contact, CodedValue, OnEnter, OnExit, OnEvent, ActivateIf, DeActivateIf
* title, enabled, visible, mustImplement, showInReport
* minCard, maxCard, repeat, instanceGUID, parentGUID

Example 9: Inherited Elements and Attributes for the XFCs that derive from RepeatingType

The following two lists show the new elements and attributes introduced by descendants of **DisplayedType** (Example 10) and **RepeatingType** (Example 11).[[34]](#footnote-35) As before, the SDC XML element name for each XFC Type is shown in parentheses after the XFC Type.

|  |  |
| --- | --- |
| **ButtonItemType** (ButtonAction):  **Elements**:  OnClick  **InjectFormType** (InjectForm):  **Elements**:  Section  Question  FormDesign  **Attributes**:  pkgID  pkgInstanceURI  pkgInstanceVersionURI  pkgBaseURI  pkgFullURI  pkgManagerUR  rootItemID | **ListItemType** (ListItem):  **Elements**:  ChildItems  ListItemResponseField  **Attributes**  Selected selectionDisablesChildren  selectionActivatesItems  selectionSelectsListItems  selectionDeselectsSiblings  omitWhenSelected  associatedValue  associatedValueType  **Response Reporting Attributes**:  repeat  instanceGUID  parentGUID |

Example 10: DisplayedType Descendants

|  |  |
| --- | --- |
| **SectionItemType** (Section):  **Elements**:  ChildItems  **Attributes:**  ordered  **Response Reporting Attributes:**  repeat  instanceGUID  parentGUID | **QuestionItemType** (Question):  **Elements**:  ChildItems  **Response Reporting Attributes:**  repeat  instanceGUID  parentGUID |

Example 11: RepeatingType Descendants

The elements and attributes introduced by each XFC will be discussed in the sections below. It is important to remember that Section and Question inherit all elements and attributes from **RepeatingType**, whereas ListItem, ButtonAction and InjectForm inherit from **DisplayedType**.

It is also important to note that only Section, Question, and ListItem can nest child XFCs directly inside a ChildItems element. DisplayedItem and InjectForm do not support ChildItems XFC nesting, although InjectForm can support descendants by injecting XFCs directly under the InjectForm element.

We now address the elements and attributes introduced at each level of SDC Schema hierarchy

## The BaseType (abstract)

The BaseType is an abstract Type that is used to define the most basic features of every element defined in the SDC Schema, including elements derived from W3C datatypes. All of the BaseType attributes are *optional* to use in SDC XML.

The BaseType attributes are explained below:

* name: A unique identifier of W3C type “ID,”[[35]](#footnote-36) which must be unique within an XML document and must adhere to W3C NCName restrictions. It is similar to a unique name for a variable, control or object, used to provide the ability for programmatic manipulation of an element. It is typically assigned by the form designer or generated algorithmically. The value of name must be unique within an FDF and FDF-R, even when FDF sections are repeated in the XML.   
    
  To avoid problems when using name attributes with programming languages, form designers should restrict to begin with a letter or an underscore and to only contain characters that are legal for variable names. These generally include letters, numbers and underscore. The use of other characters may result in errors, depending on the programming language employed. Since XML is case-sensitive, attribute values should not be made unique solely on the basis of alphabetic case, since many programming languages are case insensitive and cannot distinguish names based only on case differences.
* type: The type attribute may contain one or more custom metadata "tokens" for the element, chosen from a standardized list of terms. It uses the W3C NMTOKENS type, and thus supports multiple space-separated values. The list of terms (tokens) must be defined for each use case and are not defined explicitly by the SDC Schema model or the IHE SDC Profile.

Tokens are short alphanumeric text strings, defined by the W3C Schema NMTOKEN specification, that are defined in an use-case-specific Implementation Guide. The type NMTOKEN represents a single string token. NMTOKEN values may consist of letters, digits, periods ( . ), hyphens ( - ), underscores ( \_ ), and colons ( : ). They may start with any of these characters.[[36]](#footnote-37) type tokens may be specific for one or more kinds of SDC elements. Multiple tokens in the type attribute should be separated by whitespace.

Type tokens may be used to specify special handling by an application, and are usually used to define form display constraints, but may include other custom metadata as well. Style metadata should generally be handled with styleClass rather than type. Some type token examples include: tooltip, statusLineText, etc. In general, type metadata should not affect the information content of a form.

* styleClass: Holds developer-assigned class names for display styling, generally for use with an external style sheet. It may hold multiple Like type, it uses the W3C NMTOKENS XML type, and thus supports multiple space-separated values. Examples could be alignTopLeft, thickSection, thinSection, align:bottom, pageBreak-after, etc.
* order: A decimal attribute that allows the form template developer to define a sequential order for elements in a template. This serves the purpose of providing a definitive/original order to sections, questions, answer choices, etc., when required for display purposes. This is important when the original XML ordering may become disrupted due to the use of an implementation technology that does not natively support ordering (e.g., object collections), and it can also provide a check on the proper importing of the XML tree during implementation.

## The ExtensionBaseType (abstract)

The ExtensionBaseType (EBT) is an abstract type that confers on descendants the ability to add custom Extension, Comment and Property elements. Most SDC Types inherit from ExtensionBaseType, with the notable exceptions of CommentType, ExtensionType, all SDC datatypes, and some direct derivatives of datatypes.[[37]](#footnote-38)

### The EBT Property Element

Property elements are based on PropertyType. PropertyType, which is a component of EBT, also derives recursively from EBT. Thus, a Property element may have any number of direct Comment, Extension and Property descendants, and Property elements may thus be nested to any desired degree. Property elements may occur as children of all XFCs and most other FDF elements.[[38]](#footnote-39)

Properties may be used to define standard or custom FDF metadata. They also may be used to record visible or hidden metadata for any purpose. Property elements, when used on a DI or descendants of DisplayedType, can be used for many kinds of tasks including:

* Display static information on forms
* Display context-sensitive information (e.g. tooltips, status bar text, help pop-ups, or special text designed for report output). (The Link element may also be used for to provide help resources)
* Provide rich title text (i.e., in HTML format) for implementations that support rich text
* Provide alternative language text
* Provide ancillary, alternate, instructional or informational text for DisplayedType descendants

In most cases, Property-derived text is displayed only under certain conditions (e.g., for rendering tooltips or report output). Determining if/when a Property should be displayed on a DEF requires the interpretation of its propName, type and styleClass attributes, as specified by an established user community. Examples of possible Property propName values include “helpText”, “tooltip”, “statusBarText”, etc., which are commonly supported concepts and control types in most DEF programming frameworks. The Property element is also commonly used to contain invisible form metadata, e.g., versioning, source references, alternate language text, etc. Examples include “alt-text”, “reportText”, “htmlTitle”, etc, which will be described later. Any SDC Property text may optionally be available as strongly-typed data (e.g., integer, string, etc.).

Property may have descendants of Property, Comment and Extension. As shown in Example 12: Nested Properties, Properties may be nested to any depth and use any datatype for the Property’s value.[[39]](#footnote-40)

**<DisplayedItem** ID="DI1" title="?This is a Note"**>**  
 **<Property** propName="myPropName1" val="This is property value 1"**>**  
 **<Property** propName="myPropName1.1" val="This is property value 1.1"**>**  
 **<Property** propName="myPropName1.1.1" val="This is property value 1.1.1"**/>**  
 **</Property>**  
 </Property>

**<Property** propName="myPropName2" val="This is property value 2"**>**  
 **<Property** propName="myPropName2.1" val="This is property value 2.1"**>**  
 **<Property** propName="myPropName2.1.1" val="This is property value 2.1.1"**/>**  
 **</Property>**  
 **</Property>**  
**</DisplayedItem>**

Example 12: Nested Properties

Properties will be called by their propName, so that if the propName value is “myPropName1,” the Property will be called the “myPropName1 Property.” A Property must have content in the propNameattribute, unless it is using a strongly-typed value under the Property\**TypedValue** tag. Strongly-typed Property values are discussed next.

### FormDesign\Property elements

The Property element may be used in conjunction with the FormDesign element, as shown below:

**<FormDesign**   
 order="0" ID="Lung.Bmk.227\_1.001.011.RC1\_sdcFDF"  
 baseURI="cap.org"  
 fullURI=  
 "\_baseURI=cap.org&amp;\_lineage=Lung.Bmk.227&amp;\_version=1.001.011.RC1&amp;\_docType=sdcFDF"  
 filename="Lung.Bmk.227\_1.001.011.RC1\_sdcFDF.xml" lineage="Lung.Bmk.227"  
 formTitle="Lung Biomarker Reporting Template" version="1.001.011.RC1"  
 xmlns="urn:ihe:qrph:sdc:2016"  
 **<Property** name="Copyright" type="CAPeCC\_static\_text" styleClass="copyright" order="1"  
 propName="Copyright"  
 val="(c) 2018 College of American Pathologists. License required for use."**/>**  
 **<Property** name="ApprovalStatus" type="CAPeCC\_meta" order="13" propName="ApprovalStatus"  
 val="RC1"**/>**  
 **<Body** name="Body" order="14" ID="Lung.Bmk.227\_1.001.011.RC1\_sdcFDF\_Body"**/>**   
**</FormDesign>**

Example 13: **FormDesign**\**Property**

#### Strongly-Typed Property Values for Special Purposes

Sometimes, a Property needs to use a specific datatype, and this datatype ideally should be validated in the SDC XML for the correct format required by that datatype. In these cases, we can use the **TypedValue** feature of the Property element.

In the SDC Schema, HTML and XML are regular datatypes, in addition to the standard W3C datatypes, such as string and integer. However, HTML in the SDC Schema means the strict XHTML[[40]](#footnote-41), [[41]](#footnote-42) variant. SDC thus supports custom HTML and XML islands everywhere that Property and datatypes are supported in the FDF.

Example 14 shows a strongly typed Property used to include XHTML-formatted text at any point in any EBT-derived element. The term “strongly-typed” in this context means that we are using a well-known (and preferably well-documented) **propName** value, which in this example is “myHtmlProperty”. Note the inclusion of a required XHTML Schema xmlns namespace declaration and SchemaLocation in the first **<div>** element. They are both required for the use and validation of a Property with strongly-typed XHTML content.[[42]](#footnote-43) As noted above, the **HTML** datatype element is used for this purpose, but the validation is accomplished with an XHTML Schema, to make it produce only valid XML, and make it work with strict XML validation technologies.

**<Property** **propName**="myHtmlProperty" val="This is plain property text"**>**  
 **<TypedValue>**  
 **<HTML>**  
 **<div** **xmlns**="http://www.w3.org/1999/xhtml" xsi:**schemaLocation**="http://www.w3.org/1999/xhtml xhtml.xsd"**>**  
 This is the **<b>**XHTML**</b>** version of the property text.  
 **</div>**   
 **</HTML>**  
 **</TypedValue>**  
**</Property>**

Example 14: **Property** using **TypedValue**\**HTML**

Strongly-typed Properties support all SDC datatypes. The next example shows a strongly-typed date Property, which enforces the proper date format. Note that val is not included in the Property element, only in the date element. This is because val can only be validated against the string datatype, whereas date validates using the W3C **date** datatype.

**<Property** propName="myDateProperty"**>**  
 **<TypedValue>**  
 **<date** val="2019-01-01"**/>**  
 **</TypedValue>**  
**</Property>**

Example 15: **Property** using **TypedValue**\**date**

#### The **reportText** Property

The following example uses a Question XFC to introduce the reportText Property. We will cover the Question XFC in more detail later. Question text used for the DEF display is contained in the title attribute. In the default case, this Question title text should also appear in a report with no changes. In some cases, however, the report text should be different than the DEF text. In this latter case, the Question title is used for display to the DEF user, but the val content (**"Gross Appearance:")** of the reportText Property is used for the report.

**<Question** ID="Q6" title="Describe the Tumor's Gross Appearance"**>**  
 **<Property** **propName**="**reportText**" **val="Gross Appearance:"/>**   
 **<ResponseField>**  
 **<Response>**  
 **<string** val = "The tumor was of ovoid shape, fully encapsulated..."**/>**  
 **</Response>**  
 **</ResponseField>**  
**</Question>**

Example 16: **Property** using reportText

Occasionally, we wish to delete the DEF title text entirely from the report. This is achieved by placing “**{no text}**” in the reportText Property as follows:

**<Question** ID="Q6" title="Describe the Tumor's Gross Appearance"**>**  
 **<Property** propName="reportText" **val**="**{no text}**"**/>**   
 **<ResponseField>**  
 **<Response>**  
 **<string** val = "The tumor was of ovoid shape, fully encapsulated..."**/>**  
 **</Response>**  
 **</ResponseField>**  
**</Question>**

Example 17: **Property** using “{no text}”

The reportText Property is useful on most XFCs, but especially for Question and ListItem text.

#### Other Property Types

As described earlier, it is also possible to use a special Property (e.g., “**titleHTML**”) to specify rich text for the DEF and/or report:

**<Question** ID="Q6" title="Describe the Tumor's Gross Appearance"**>**  
 **<Property** **propName**="**titleHTML**"**>**  
 **<**TypedValue**>**  
 **<HTML>**  
 **<div** xmlns="http://www.w3.org/1999/xhtml"  
 xsi:schemaLocation="http://www.w3.org/1999/xhtml xhtml.xsd"**>**  
 Describe the Tumor's **<b>**Gross Appearance**</b>** **</div>**  
 **</HTML>**  
 **</TypedValue>**  
 **</Property>**  
  
 **<Property** propName="reportText" val="Gross Appearance:"**/>**  
 **<ResponseField>**  
  **<Response>**  
 **<string** val="The tumor was of ovoid shape, fully encapsulated..."**/>**  
 **</Response>**  
 **</ResponseField>**  
**</Question>**

Example 18: Rich text using **Property**\**TypedValue**\**HTML**

The Response text can handle rich text from user-input in a similar fashion, but *without* using Property. For example, the R**esponse\string** datatype could have used the **HTML** datatype instead of **string**, allowing the DEF user to enter rich HTML text into the DEF instead of unformatted text:

**<Response>** **<HTML>**  
 **<div** xmlns="http://www.w3.org/1999/xhtml"  
 xsi:schemaLocation="http://www.w3.org/1999/xhtml xhtml.xsd"**>**  
 **<b>**The tumor was of ovoid shape, fully encapsulated**</b>**   
 **</div>**  
 **</HTML>  
</Response>**

Example 19: Using **Response**\**HTML** text

The next example demonstrates how a Property can define text that appears in a report, but should not appear in a DEF. By default, DisplayedItem title text should not appear on a report, because it is usually designed to aid the DEF user in filling out the DEF. However, custom text is sometimes needed on a report, but is not needed (by default) on the DEF because it can clutter up the screen. This pattern is sometimes called a “report note.” Note that the title attribute could have been omitted form the XML entirely, since it’s an optional attribute that contains no value ("").

**<**DisplayedItem ID="DI2" title=""**>**  
 **<Property** propName="reportText" val="Physician's Note: ..."**/>**  
**</DisplayedItem>**

Example 20: Using the reportText **Property** on a **DisplayedItem** for reporting notes

Later, we will introduce the concept of “untitled Questions,” which are Questions that have no title attribute text, but the Question’s subject is easily inferred from the context. However, in some cases, a particular DEF implementation style may need to display alternative text for the question. For these situations, alternative title text is provided in the form of the altText Property, as shown below:

**<Question** ID="Q7" title=""**>**  
 **<Property** propName="altText" val="This is alternative Question text"**/>**

<!-- other Question XML elements omitted -->  
**</Question>**

Example 21: The altText **Property**

Properties can be used for any similar custom purposes on any EBT element. The Property names (propName) and implementation details must be agreed upon by form designers and implementors so that implementation code.

### The EBT Comment Element

The EBT also includes the ability to add optional Comment elements. The Comment element is defined by the CommentType, which inherits from BaseType. Comments may be provided by the FDF designer, the FDF implementation code, or by the DEF user. The type and styling of the Comment (if present) can be customized with the type and styleClass attributes. The comments are simple ASCII text and may not include rich text or substructures. The Comment content is placed in the val attribute of the Comment element.

Examples of comments created by an FDF designer are comments for the implementor. Comments from the DEF *user* might concern their response to a QR or the selection of a ListItem. If DEF users are allowed to add Comment elements, then the FDF implementation must include visual clues (e.g., a comment icon) in the DEF so that the user can add comments at appropriate points (e.g., on each Question and ListItem).

### The EBT Extension Element

The Extension element provides a place to insert XML that is not defined in the SDC Schema. The Extension element is defined by ExtensionType, which inherits from BaseType. The sub-elements of Extension must provide a namespace (and ideally an XML Schema) for any non-SDC elements and attributes that are introduced. The Extension element provides almost infinite expansion flexibility for SDC. However, it also requires that form designers and implementers agree on the supported extensions, agree on which elements and contexts they may be used, document this usage, and create an extension validation mechanism to enforce correct usage and detect incorrect usage.

## IdentifiedExtensionType (abstract):

The IdentifiedExtensionType (**IET**) adds two attributes to its parent Type, EBT. The two attributes allow the unique identification of XFCs in an FDF, and also allow unique identification of the FDF. The attributes are:

* ID: The ubiquitous ID attribute is a unique URI identifier for XFC types in FormDesign, and for FormDesign itself. It is required, and its uniqueness is enforced by the SDC Schema. URI identifiers are very flexible since they may assume any legal XML URI format.
* baseURI: The baseURIis required only in the FormDesign element; it is optional on XFCs. It identifies the organization that is responsible for designing and maintaining the FDF or XFC. If an XFC does not derive from the same organization as the default baseURI (the FDF baseURI which is defined in the FormDesign element), then a new value for the baseURI is entered on the XFC element to override the default one, and the new baseURI is then inherited by all descendant XFCs unless overridden by a descendant XFC. In most cases, FDFs will contain only a single default baseURI on the FDF, and no baseURI on the XFCs
* Ideally, the baseURI + ID should combine to form a globally unique identifier, that uniquely identifies an item in a particular form lineage. The same baseURI and ID may be reused in derived or versioned forms, as long as the context stays the same, and any affected data elements remain unchanged in context and semantics. See section ‎4.3.3 for more information.

## DisplayedType

DisplayedType has two functions: it defines the DisplayedItem XFC for representing visible areas on the screen, and also acts as a building block for the other visible XFCs (Section, Question, ListItem, ButtonAction). DisplayedType defines most of the essential functional underlying capabilities of XFC-based controls. These capabilities include support for LinkItem, BlobContent, CodedValue, and several fundamental **Event** and **Guard** types. LinkItems contain the address of internal (DEF-based) or external (network-based) resources for support display or functionality in a DEF. BlobContent contains inline Binary Large Objects of virtually any type, but primarily those defined as standard MIME types. **Events** are DEF user actions that are “captured” by the DEF to alter DEF behavior or functionality in some way. The standard events are Enter, Exit, OnEvent (designer-defined). OnEvent is a generic event that is defined by the form designer. An SDC **Guard** is a unit of coded functionality that is activated or “fired” by an Event somewhere else in the DEF. The two built-in Guards are ActivateIf and DeActivateIf. These Guards activate and deactivate visible XFC-derived DEF controls based on activity somewhere else in the DEF.

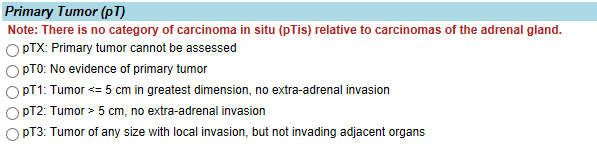
## RepeatingType

# The XFCs

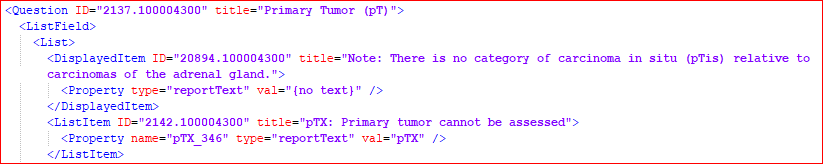
## XFC Identifiers and Names

## The DisplayedItem XFC

As described above, DisplayedType, which provide the DisplayedItem (DI) definition, is the parent Schema Type of the XFC Types for S, Q, LI, and BA. Thus, S, Q, LI, and BA inherit all elements and attributes of DisplayedType. IF does not inherit from DisplayedType, as it relies on the display properties of the subsumed injected parts, which do inherit from DisplayedType.

The primary function of DI is to provide text in the DEF. Often this takes the form of “notes,” which are strings of text that may accompany QAS controls to provide explanatory information with background documentation. Although DEF notes usually derive from the DI XFC, they can also derive from other SDC objects such as Section titles and Property values. The DI (as well as all DisplayedType descendants) can additionally be used to display Links (e.g., to Internet resources), BlobContent (Binary Large Objects) of any MIME type, Contact information and CodedValues.

DisplayedItem as a **Note**



**DisplayedItem**: functions as a note inside a list of answer choices

A DisplayedItem has a unique ID and a title, but unlike Questions and Sections, cannot have ChildItems form components. DisplayedItems that must have XFC descendants should use the Section model.

### DI Substructure

#### Blob

Blobs can handle any type of binary media, such as images, audio, video, and data streams of any specified format (such as a MIME type). The blob may contain any type of special media but must be base 64 encoded.

The generic XPath for Blobs is DisplayedItem/BlobContent. Blob children elements are Hash, BlobURI and BinaryMediaBase64.

#### Link

Describe DEF display handling of Link: inline, pop-up, model, new window/tab, etc.

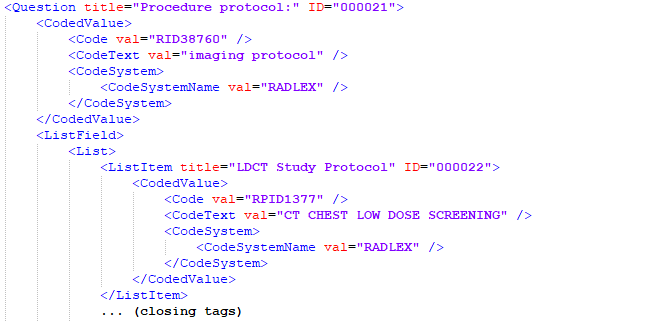
#### CodedValue

The CodedValue element provides an alternate hidden metadata encoding a code, terminology, classification, keyword, or local value that may be necessary in certain use cases. It has the following children elements:

* **Code**: A standard code, or a local value from a custom coding system, that can be used to consistently identify, or provide a standard value for, the coded item.
* **CodeText**: The human readable text that accompanies the assigned code and represents the code's precise meaning (semantics) or usage.
* **CodeMatch**: Degree of match between the mapped item and the assigned code. Its codeMatchType attribute holds an entry from an enumerated list of match types as follows:
  + Exact Code Match
  + Close Code Match
  + Code Broader Than Item
  + Code Narrower Than Item
  + Item Implements Data Element Exactly
  + Item Derived from Data Element
  + Item Related to Data Element
* **CodeMatchComment:** Comment about the degree of match between the mapped item and the assigned code.
* **CodeSystem**: The parent element whose children define the system that creates and maintains the standards for the code map. Its child element is CodeSystemName**.**
  + **CodeSystemName**: The name of the coding system, as recommended by the coding system curators, or as recommended by the agency that creates standards for the code map in use.
* **ReleaseDate**: The day that the selected version of the coding system was released for general use by the coding system curators.
* **Version**: Version of the coding system; uses the version format defined by the coding system.
* **OID**: The ISO object identifier (OID) for the coding system, as found at the HL7 OID Registry: https://www.hl7.org/oid/index.cfm
* **CodeSystemURI**: Web resource that uniquely identifies the coding system.

The element **DefaultListItemDataType** may also be used with code systems to enumerate a default data type if LIs are all associated with coded values from a single coding system.

vents and Guards



CodedValue

segment

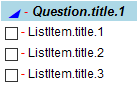
CodedValue segment

## The Section XFC

**Section** controls derive from FDF Section elements. They contain an ID and title and they subsume (wrap) a group of topically-related form controls. Sections can contain any number of sub-sections and other XFCs, except for ListItems. ListItems may appear in a Section but must be within a Question (QAS) structure.

## The Question XFC

Each QAS in the FDF XML is an XFC that can be represented in the DEF by one or more controls (widgets) by the FF software. The controls capture user response data in the DEF. Each QAS XFC has an ID [[43]](#footnote-44) and other metadata. DEF controls are assigned the matching XFC ID to enable data from the DEF control to be captured into the XFC XML that shares its @ID. The behavior of the control is determined by the corresponding XFC metadata. The primary visible text for all controls is found in the title attribute. The following figure depicts a simple multi-select Question (QM)

**<Question ID="QM1" title="This is a QM Question title" >  
 <ListField maxSelections="0">  
 <List>  
 <ListItem ID="LI1" title="ListItem.title.1" />  
 <ListItem ID="LI2" title="ListItem.title.2" />  
 <ListItem ID="LI3" title="ListItem.title.3" />  
 </List>  
 </ListField>  
</Question>**

Example 22: Question XFC XML

The three basic QAS types will be described next: Question-Response, Single-Select Question, and Multi-Select Question

### Question-Response (QR)

The **QR** XFC does not have an answer list (a List of ListItems). **[[44]](#footnote-45)** The QRcontrolrepresents a Question control with an adjacent area for capturing a **r**esponse from the user.

A QR response may be validated by rules in the Response element, which determine if the inputs are in the proper format (e.g., as a text string or decimal, or rarely, following a pattern mask or a binary type).

The following FDF XML sample contains the basic parts of a QR XFC:

**<Question ID="40273.100004300" title="Comment(s)">  
 <ResponseField>  
 <Response>  
 <string maxLength="4000" minLength="0"/>  
 </Response>  
 </ResponseField>  
</Question>**



**QR**

**ResponseField**

Example 23: QR XFC XML

Observe the following points about this sample:

* Question is the only XFC that appears in this XML example. The other nested elements support the Question XFC with additional metadata.
* All XFC elements (Question in this case) must have an **ID** that is unique within the FDF which enforced by the XML Schema. Uniqueness of IDs must be confirmed by SDC-compliant software. Since XFCs can be repeated in DEF implementation, uniqueness-enforcement is an important task in SDC software implementations.
* The text to display in the DEF is in the **title** content
* Responses are captured in a ResponseField / Response / *(datatype)* structure, where *(datatype)* is an SDC element corresponding to one of the SDC datatypes.[[45]](#footnote-46)
  + Responses must contain an SDC datatype element (string in this case)
  + User-entered responses are captured in the **val** attribute of the *(datatype)* element, like this:

**<string maxLength="4000" minLength="0" val="This is my comment" />**

#### The ResponseField Element

(ListItemResponseField too)

##### The Response Element and its Datatypes

* All datatype elements names use lower camel case,[[46]](#footnote-47) unlike all other elements in the SDC Schema. This is done to conform with the W3C representation of the datatypes in XML Schema documents.
* All datatypes are defined and modeled after the W3C datatypes and include all of the W3C datatype metadata attributes.
* All metadata attributes have a type appropriate for the element and the metadata. For example, **maxLength** has an XML Schema datatype of xs:long[[47]](#footnote-48).
* The **val** attribute validates the datatype of its element tag. Entering an invalid datatype in this attribute will generate a validation error when validated with the SDCFormDesign Schema.
* **val** cannot be Schema-validated against the element’s other datatype metadata (for **string**, in this case, **maxLength**, **minLength**, **mask**, **pattern**), so that further validation of **val** content must use additional methods such as Schematron.

Here is another example that shows how numeric QR structures may appear:

**<Question ID="QR1" title="Specify Percentage of QR quarks (see Note A)">  
 <ResponseField>  
 <Response>  
 <integer maxInclusive="100" minInclusive="1"/>  
 </Response>  
 <TextAfterResponse val="%"/>  
 <ResponseUnits val="%" unitSystem="UCUM"/>  
 </ResponseField>  
</Question>**

Example 24: QR with Numeric ResponseField

Observe the following points:

* This example uses an <**integer**> datatype to specify and record the user-entered value.
* When the user enters a response into the DEF, the <**integer**> element will be populated with val and look like this:

**<integer maxInclusive="100" minInclusive="1" val="My user response"/>**

* The displayed datatype validation metadata are specific for the **integer** datatype (**maxInclusive**, **minInclusive**).
* Other attributes are available for numeric response datatypes, including: **quantEnum**, **maxExclusive**, **minExclusive**, **totalDigits**, **mask**, **allowGT**, **allowGTE**, **allowLT**, **allowLTE**, and **allowAPPROX**. For decimals and float, , **fractionDigits** is also available.
* The **TextAfterResponse** element provides a place to specify static text that should appear *after* the user’s response in the DEF and the report. The **val** content holds the text for display.
* The **ResponseUnits** element provides a place to record the units relevant to the response. The **val** attribute holds the text ("%") of the units for the user’s response. This is provided by the FDF designer, not the DEF-user. The **ResponseUnits** is “hard-coded” into the FDF and is not changeable via the DEF. The **unitSystem** attribute default[[48]](#footnote-49) to “UCUM” (Unified Code for Units of Measure) and is not usually displayed in the DEF or report.

### Single Select Questions

A **QS** control displays a Question that subsumes a List of ListItem controls. The List is set to single-select in the XFC metadata and in the DEF control. The QS control is often rendered as a Question title area along with option/radio buttons for the LIs, or as a combo-box with dropdown answer choices, from which only one answer may be selected. In the XFC, the QS ListField element has maxSelections="1" which is the default value for this attribute; if maxSelections does not appear in the ListField (as in the example below), then the Question is treated as a QS.

**<Question** ID="QS1" title="Select your favorite color for QS"**>**  
 **<ListField>**   
 **<List>**  
 **<ListItem** ID="LI1" title="Baseball"**/>**  
 **<ListItem** ID="LI2" title="Zebra"**/>**  
 **<ListItem** ID="LI3" title="Snort" selected="true"**/>**  
 **<ListItem** ID="LI4" title="Zinger"**/>**  
 **<ListItem** ID="LI5" title="Enter another color" **>**  
 **<Property** propName="reportText" val="{no text}"**/>**  
 **<ListItemResponseField** responseRequired="true"**>**  
 **<Response>**  
 **<string** maxLength="30" **/>**  
 **</Response>**  
 **</ListItemResponseField>**  
 **</ListItem>**  
 **</List>**  
 **</ListField>**  
**</Question>**

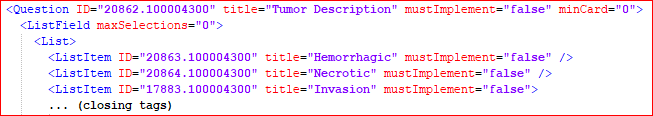
**ListField**: maxSelections="1" is not shown, since it is the default value

Example 25: QS XFC

#### The ListItemResponseField

### Multi-Select Questions

A **QM** control also displays a Question that subsumes a List of ListItems. This ListField element that subsumes the List element is set to multi-select by setting maxSelections to 0 (no limit on number of selections), or to a value greater than 1 (if there is a limit to the number of selections allowed). The control is often rendered with *checkboxes* for LIs, from which more than one ListItem may be selected. These checkboxes may be part of a combo-box/dropdown control as well.



**ListField**: @maxSelections = “0”, but QM also could have any number greater than 1



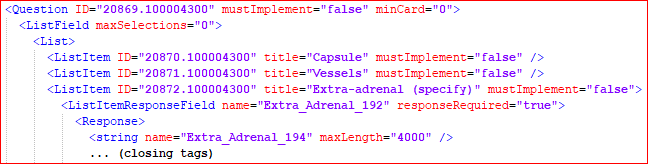
As described earlier, there are 2 ListItem types:

Example 26



* An **LI** control that displays a simple ListItem, as described above
* An **LIR** control displays a ListItem with an associated visible Response field to capture the user’s fill-in response, similar to a QR.

Example 27



**LIR**

### Capturing User Responses in Questions

Capturing data in a Question can occur in two ways:

* **Response**: A response (answer) to a QR or LIR is captured directly into the FDF Response element in the @val attribute:

QR

<Question ID="q2" title="My Question">  
 <**ResponseField**>  
 <**Response**>  
 <**string** **val="My string value"**/>  
 </Response>  
 </ResponseField>  
 </Question>

Example 28

LIR

<Question ID="q1" title="Question-1">  
 <ListField>  
 <List>  
 <ListItem ID="341" title='ListItem-1' selected="true">  
 <**ListItemResponseField** responseRequired="true">  
 <**Response**>  
 <**string** **val="My string response"** />  
 </Response>  
 </ListItemResponseField>  
 </ListItem>  
 …  
 </List>  
 </ListField>   
 </Question>

Example 29

* **Selection:** Each selected ListItem on a QS or QM is captured by setting the selected attribute on the ListItem element to ‘true’. If the selected attribute does not appear, it takes its default value, which is ‘false’.

<Question ID="q1" title="Question-1">  
 <ListField>  
 <List>  
 <ListItem ID="341" title='ListItem-1' **selected="true"**/>  
 </List>  
 </ListField>   
 </Question>

Example 30

The **FDF** may contain Question items that are pre-configured to contain *default responses* by setting selected to “true” or providing Response data in the val attribute. These default responses are set as illustrated in the prior bullets and are displayed in the DEF upon loading the form.

### Response Metadata

## DEF Helper Components

In addition to QAS controls, the FDF can define ancillary viewable or structural components such as DisplayedItems and sections, as well Property metadata that can be made viewable in a DEF. Helper components organize a form’s content and/or provide additional textual or media information or cues. The three most common types of helper components are presented below:

# DEF Functional Considerations

## The DEF Maintains and Manipulates the FDF

The DEF, running inside a FF application, must maintain a copy of the FDF and manipulate its XML to add the user’s responses, captured by the DEF, into the XML. This process involves inserting the captured responses into specific XFCs in the FDF. FDF XML manipulation can occur with each user interaction and/or at the time of form data submission to the Form Receiver.

## Implied Activation (IA)

An “activated” DEF item means that a user can interact with that item. If the active item is a Question, then the user can answer it. Thus, an active item is both *visible* and *enabled*.

Conversely, a *de*activated item is disabled and/or invisible, so that a user cannot interact with it in the DEF. If an item is deactivated, then all its descendants are also deactivated. (The only exception is provided with selectionDisablesChildren, described later.)

A deactivated item must not be subject to answer validation, should not appear on a report, and should not be saved in a database, even if it contains a response from earlier user activity in the form. Such earlier activity in a previously-activated item is now overridden by the deactivated status of the same item.

The hierarchical arrangement of items in the FDF file, as reflected in the DEF, allows us to define a drill-down item activation pattern as the user interacts with DEF items. This drill-down behavior can affect the layout, workflow, validation, reporting and data storage properties of DEFs.

In the Example 30A below, ListItem LI.1b is the parent of **QS2** in the XML and in the DEF. **QR3** is the parent of **QS4.**

The DEF will open with **QS1** and **QR3** activated, but **QS2** and **QS4** inactive/disabled.

In general, when opening an eCC DEF for the first time, only the outermost level of the DEF item tree should be activated.[[49]](#footnote-50) As the user answers Questions, new parts of the item tree become activated. This “Implicit Item Activation" occurs when a selected ListItem or an answered QR activate successive layers of dependent (child) items.

If LI.1b is selected, then **QS2** will become activated. When **QR3** receives a fill-in response, then **QS4** will be activated.

If the parent **QS1** and **QR3** responses are removed, then the activation of the child items is reversed (they become deactivated again). If the parent **QS1** and **QR3** items become deactivated by *their* parent items, then all their descendants (including **QS2** and **QS4**) also are deactivated.

|  |  |  |
| --- | --- | --- |
| **Implied Activation (A)** | **Implied Activation (B)** | **Key for Examples:**  **Green = Parent Question**  Blue = 🞏 ListItem (multi-select ListItem)  ⭘ ListItem (single-select ListItem)  **Red = Child (dependent) Question** |
| **QS1:**   * LI.1a * LI.1b   **QS2**   * + - LI.2a     - LI.2b   **QR3: \_\_\_\_\_\_\_\_**  **QS4**   * + - LI.4a     - LI.4b | **QS1**   * LI.1a * LI.1b   **QS2**   * LI.2a * LI.2b * LI.2c |

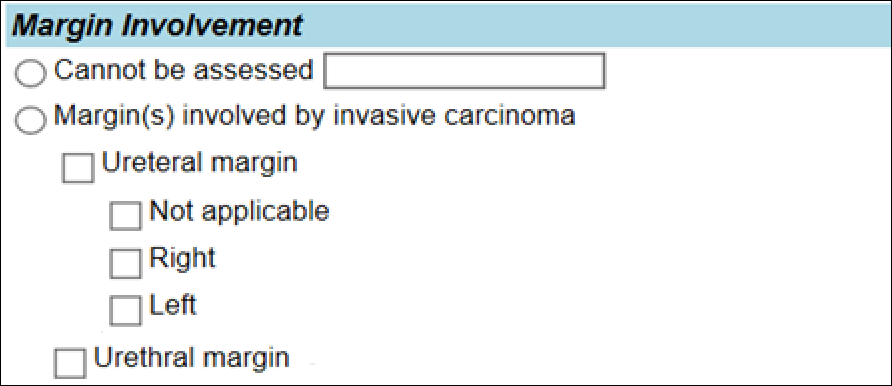
Example 31: Implicit Item Activation

In Example 30: Implicit Item ActivationB, QS2 is a direct child of QS1. In other words, QS2 is not a child of LI.1b. Thus, QS2 is activated when any ListItem (LI.1a or LI.1b) is selected from QS1.

The SDC model works well when IA is implemented in the DEF using appropriate enabled/disabled or visible/invisible properties.  If a parent item is deactivated, then that all descendant items must also be deactivated. User-entered answers for deactivated items should not be present in the stored dataset.

### Implied Activation with Complex Nesting and Invisible Sub-Questions

In the following figures, observe that Questions QM1 and QM2 are nested under parent ListItems (LI2 and LI3). QM1 and QM2 only become activated[[50]](#footnote-51) if their parent LIs are selected.This is an example of IA, described above. When rendered as form controls, the child XFCs (and all their descendants) are inactive/disabled until the parent Question response is *captured*. A QR response is captured by entering a response (fill-in) value in the DEF. A QS or QM ListItem selection is captured by selecting one or more LIs. The figure above depicts a *complex* QAS (a QAS with descendant Questions) showing multi-level control nesting that mirrors the FDF XML form component nesting.



**QS**

**LIR**

**QM1** (no **@title**) (inactive unless LI2 is selected)

**LI22**

**QM2** (no **@title**) (inactive unless LI3 is selected)

**LI3**

**List** (inactive unless LI3 is selected

**LI12**

Example 32: DEF with complex nesting and IA

Visibility of inactive controls is subject to personal preference and usability considerations. In general, the visibility of inactive controls is left to the FF design and usability team (aka, the user experience or “UX” team) to decide.

Note that QM1 and QM2 have no title value in the XML, but the ListItems do have title values. These Questions with no @title value are often called **invisible Questions***.* Although no QM1 or QM2 Question text is visible in the DEF, the nested ListItems provide a hint to the existence of the 2 invisible Questions, and the Questions’ meaning can be readily inferred from the QAS context.

In most cases, inactive or read-only status should be rendered as grayed-out disabled controls. However, consider that hiding and unhiding controls [changing visibility] can be distracting to users, so activation/deactivation behavior must be carefully evaluated.



**LI1**

**QM2**

**LI3**

**QM1**

**List**

**QS**

**LIR**

**LI2**

Example 33: SDC XML with complex nesting and IA

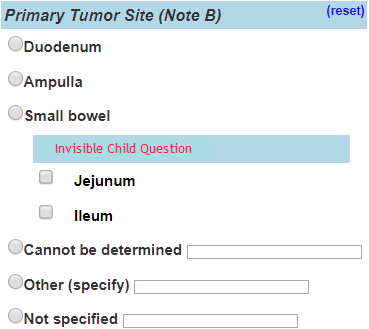
### Complex Item Dependencies

Although IA covers most simple dependencies in SDC, some use cases may require that Questions become activated or deactivated based upon more complex criteria.  For example, Question 4 may be come activated when both Question 1 and Question 2 contain specific values, and Question 3 does not contain either of two specified values.  These relatively complex cases will be addressed in later sections.

## Invisible Question Text

As noted above, missing title values in a Question results in the creation of “invisible Questions,” where ListItems are visible, but the parent Question text is not. Blank titletext is used to unclutter the DEF when the meaning of “sub-answers” is obvious from the context, and Questions title text would be distracting. However, several some implementers have requested that the FDF suggest *alternative text* that reflects what the Question text could be, if it were displayed in the DEF. Therefore, the FDF files supply can these inferred terms as a Property with propName = “altText” for invisible Questions. However, the **altText** Property values may have a deleterious effect when the DEF authors have decided that Questions title text is best omitted. Importantly, **altText** Propertyvalues are not necessarily appropriate for use in synoptic report output. In general, **reportText** Property values should be used to customize report output.

In some cases, the **altText** Property value merely replicates the titlevalue from the parent Question, because the parent Question text continues to be appropriate for the child ListItems. For example:



Example 34: Invisible Question

In this case, the **altText** Property value for the *Invisible Child Question* may be assigned the same text value as the parent Question (val= “Primary Tumor Site”), or it may be assigned a more specific descriptive value, such as “*Small Bowel Site*.” As a general guide, **alt-text** from invisible Questions should be suppressed whenever it merely duplicates the parent Question, as in the above example.

The **altText** Property values should be created in a terse manner consistent with the original Question semantics. However, this is not possible in some cases, especially when terse wording can result in ambiguous terms.

The **altText** Property value for the invisible Questions can be found as a Property under the invisible Question. Implementers should check with their end-users to determine whether they wish to use the supplied **altText** Property value, substitute their own appropriate text, or suppress the DEF Question text entirely.

We recommend that **altText** Property values be used only for internal purposes (e.g., as a guide to the visual identification of Questions in database queries) and not be displayed as visible text in the DEF unless approved by the end-users.

## The @mustImplement Attribute

The mustImplement attribute may be found on any XFC. By default, a QAS must generally be implemented on a form, so that users can capture a response in that control. If mustImplement = “true” (the default) or missing in the XML, then the XFC must be implemented in the form.

However, if mustImplement = “false”, then the XFC need not be implemented as a control in the DEF.

### Optional ListItems

In rare cases, individual ListItems to a required Question can be optional to implement. For all optional ListItems, the value of mustImplementis “false” on the SDC ListItem element. In some cases, a plus sign (+) may be used in the ListItem’s title value in the SDC XML, and the “+” may also appear in the DEF.

## Required Responses

The appearance of a control in a DEF does not imply that it must receive a response from the user. Additional attributes are used to indicate that a response must be captured as a selected LI or as a fill-in response:

* minCard **>0 for a** Question **means that the** Question **must receive a response.**
* responseRequired=”true” for an LIR’s ListItemResponseField element means that a valid value must be entered in the ListItemResponseField.

As indicated in the figure below, the attributes that indicate optional responses are:

* minCard **= “0” for a** Question
* responseRequired is *false (or missing)* for an LIR’s ListItemResponseField element



**@responseRequired = “true”** means that, if this **ListItem** is selected, then it *must* receive a response from the user. This holds even if the Question is optional (as in this case).

**@mustImplement=”false”** means that this Question need not appear in the DEF.

**minCard = “0”** means this Question does *not* need to be answered (it is optional)

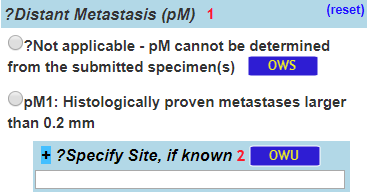
An implementer *may* also use DEF color-coding and/or other visual representations to indicate that a required item must have a response captured.

Example 35: Required responses

In some DEFs, optional sections, Questions and ListItems are prefixed with a plus (+) sign. In the SDC HTML forms, the “+” is used when minCard = “0” on Questions, and when mustImplement = “false” on ListItems.

An entire template can be optional as well, and this will be determined for each use case, such as commonly occurs with accreditation or quality assurance requirements. An optional SDC FDF does not imply that every Question will also be optional to answer (minCard = “0”). Thus, XFCs in an optional template can still be flagged as required (minCard > 0 and/or mustImplement = “true”). Once a decision is made to use the optional template, the required parts will still be flagged by the DEF validation engine.

## Conditionally Required (CR) Behaviors and Reporting from the DEF



Example 36: Conditionally Required (CR) behaviors

All XFCs that have mustImplement **= “true”** must appear in DEF so that users may complete them. However, the *responses* from some of these “mustImp” XFCs should be **omitted** from **reports** under specific circumstances.

Answering and/or reporting the contents of Sections or Questions may be required (e.g., for accreditation) only under certain conditions, and these are called ***conditionally-required*** (CR) items.[[51]](#footnote-52)

SDC Implementers should use the FDF metadata to determine the presence of CR items, as described below in more detail. Implementers should choose an appropriate visual method to flag the required, CR and optional Questions displayed in a DEF.

### Use of the “?” Prefix Display Model:

In one DEF convention, the title values of CR Items are prefixed in the DEF with the “**?**” symbol in the FDF (see above figure), so that an end-user may recognize when and how an item will be conditionally omitted from the report. Use of the “?” symbol in the DEF is not required, but alternate visual cues should be approved by the end-users. In the “?” display model, all CR Questions have a “?” prefix. The type of CR is determined based on the presence of a “?” prefix on ListItemtitle values, as described below. **“**

**?” prefixes should never be displayed in SDC reports.** When “?” is used, a reportText Property with the corrected report text should be provided in the FDF metadata for each affected XFC.

The two basic types of CR items (OWU and OWS) are described next.

### Omit When Unanswered (OWU)

Some Questions (or a Section or DisplayedItem preceding them) contain conditional words like “required only if,” “if known,” or similar language or intent. The ability to answer these conditional Questions depends on the responses to other Questions, and/or on information not explicitly present in DEF. The XFC Question metadata to describe OWU consists of: mustImplement **= “true”** and minCard **= ”0”**. This means that an OWU Question must be displayed on the DEF, and it *must be answered* if it is applicable according to the specified condition (e.g., “if known”).[[52]](#footnote-53) However, this is a judgment that can only be made by the DEF-user. This is demonstrated on Question 2 in the above figure.

If the “?” CR display model is used, the CR Question has a “?” prefix, but none of the ListItems have the “?” prefix in the title attribute.

Unanswered CR Questions are not reported, just like all unanswered Questions.

### Omit When Selected (OWS)

If an LI has omitWhenSelected **= “true”** (Question 1 is the above figure), then selection of that ListItem should result in the default omission of the entire QAS (and all descendants of the Question and all contained ListItems) from the *report*.

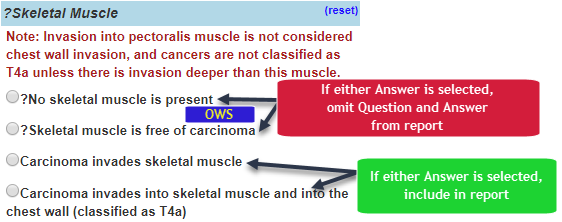
Even when omitted from the report, OWS responses must nevertheless be validated normally by the DEF software and their responses should be saved in the eCC data store (e.g., EHR database), as with any other answered Question. Storage of the non-reported data may be needed for several purposes such as quality or medico-legal review.

If the “?” display model is used for OWS, the Question and all OWS-flagged ListItems have the “?” prefix in the @title attribute. The “?” on the OWS-flagged ListItem is simply a convenient visual flag for omitWhenSelected=”true” in the SDC XML, and it informs the DEF-user that, if the OWS ListItem is selected, the QAS will be omitted from the report.

The *report-generator* must recognize the omitWhenSelected metadata, and, by default, omit the appropriate sections from the pathology report. The report generator must also allow users to override the OWS default behavior if desired, thereby displaying OWS Questions and ListItems in the report.

Unlike OWU Questions, *OWS* Questio*ns must be answered*, even if they are omitted from the report. Thus, unanswered OWS Questions should be flagged by the DEF validation mechanism to unobtrusively warn the user. However, not answering an OWS Question should not affect accreditation status. This is because accreditation reviewers look at reports, not at the stored answers - Thus, from the report alone, it is not always possible to tell the difference between an unanswered OWS Question, and one that was answered but properly omitted from the report. The limitations of OWS are recognized by the DEF authors, but this validation and reporting technique is an intentional approach to producing less verbose reports.

Example of Default Behavior for Two OWS ListItems:



Example 37: Omit When Selected (OWS)

In the above example, two ListItems (red box) are marked as OWS in the XML. When either one is selected, the Question and its responses are omitted from the report, by default.

## Contiguity of ListItem Lists

The *answer list* resides inside the Question\ListField\List element tags. Within a set of ListItems (the *answer list*, inside the ListField\List element tags), no Section, Question, ButtonAction or InjectForm may appear.  (However, these items may appear after the Question element is closed, or nested under any ListItem.) In contrast, a “List Note**”** is a DisplayedItem XFC that can appear inside a ListField\List, in any position.

The following example overlays another color scheme to highlight a few points. Green elements are legal, and *red italic* elements are illegal according to the SDC Schema. The example shows the Listelement tag which wraps the answer list for a Question. The Listtagcontains two ListItems and one DisplayedItem. The *Section* and *Question* elements are inside List and are thus illegal in the SDC XML. If *Section* and/or *Question* appeared immediately after the closing </Question> tag, they would be legal.

|  |
| --- |
| <Question>  <ListField>  <List>  **<ListItem/>**  ***<Section/> (illegal)***  **<DisplayedItem/> (This is a legal “List Note”)**  **<ListItem/>**  ***<Question/> (illegal)***  **<ListItem>**  </List>  </ListField>  </Question> |

Example 38: Contiguity of ListItem Lists

In addition, most XFCs may appear indented under a legal ListItem, but they must first be wrapped in a ChildItems element, as shown in the following example. Although most XFCs can be legally nested under a ChildItems tag, a ListItem may only be nested under a List tag, in a Question\ListField\List structure. The red *ListItem* and other tags are thus *illegal*:

|  |
| --- |
| …  <ChildItems>  <Question>  <ListField>  <List>  **<ListItem>**  ***<ListItem/> (illegal – not under <List>)***  ***<DisplayedItem/> (illegal – not under <ChildItems> or <List>)***  ***<Section/> (illegal – not under <ChildItems>)***  ***<Question/> (illegal – not under <ChildItems>)***  <ChildItems>  **<Section/>**  **<Question/>**  **<DisplayedItem/>**  ***<ListItem/> (illegal – not under <List>)***  </ChildItems>  ***<ListItem/> (illegal – not under <List>)***  **</ListItem/>**  **<DisplayedItem/>**  **<ListItem/>**  **<ListItem/>**  </List>  </ListField>  </Question>  **<Section/>**  **<Question/>**  **<DisplayedItem/>**  </ChildItems>  … |

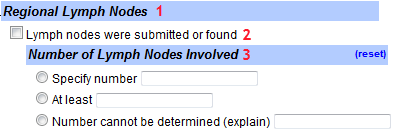
Example 39: Illegal XML structures in List\ListItem

## The Null Check Box

The SDC model intentionally does not support the concept of a 3-state (true, false, null) or “nullable” checkbox as a ListItem control, since all ListItems/checkboxes must represent a value of true (selected) or false (unselected).  Questions that require an explicit *Un*answered (Null) choice (as distinct from false) should preferably be reformulated as a Single-Select (i.e., combo box) type of Question, with mutually exclusive choices such as Present/Absent/Unknown/Cannot be Determined.

## The Single Check Box

The figure below is an example of a Question (1) with a Single Check Box (2) and Sub-Question (3)



Example 40: The single check box

By SDC modelling convention, any Question with a single ListItem must be a QM. This is because a QM is typically implemented with checkboxes that can be readily deselected to undo a previous ListItem selection. On the other hand, a QS is usually implemented as a combo box or option button paradigm, and these paradigms do not typically support deselecting ListItems.[[53]](#footnote-54) As described earlier, the QM status is set on the parent Question, setting @maxSelections = “0”.

In many cases, a QM with a single checkbox will be [*required to*](#_Required_and_Optional) *answer* (i.e., **@minCard** >0).[[54]](#footnote-55) When validating template data for required Questions, this type of Question must be scored as “answered” even when the checkbox (**2** in the example above) is not selected, since unselected (unchecked) is a valid answer choice for a single check box.

### Reporting from the Single Check Box

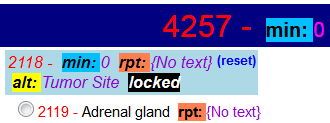
If the single checkbox (**2**) is not selected (checked), then neither the Question (**1**) nor the unchecked ListItem (**2**) should appear in the report. In normal cases, all Q1 descendants (**3**) would also be omitted from the report. This behavior (omission from the report) is expected, e.g., by form designers and accreditation agencies. However, the activation and reporting of the sub-Question (**3**) also depends on the selectionDisablesChildren (***SDAC****)* attribute on ListItem (**2**), which will be described later in detail.

### The Locked QAS

Occasionally, a QAS may need to be locked, disallowing any editing of its ListItems or response values by the end user. This technique is used in conjunction with the provision of default values, such as selected **= ”true”** on LIs, or default responses inside a QR ResponseField or the ListItemResponseField on a LI. The locked status is expressed by setting the Question’s readOnlyvalue to “true”.[[55]](#footnote-56) A locked QAS block is often included in an FDF, e.g., to represent a body structure or specimen that is always present, by definition, for a particular type of template.  These items are included in the FDF to facilitate searching of the stored data.

The locked, non-editable, information may be optionally displayed on the DEF, but end-users frequently prefer that locked items are hidden. If it is displayed, it should not be represented in the form of a selectable control (e.g., combo box).  To facilitate a consistent approach to querying on common data elements, the locked information should be stored in the SDC data store (e.g., database), along with the other Question/Answer pairs.

Most locked Questions will have the value of mustImplement = true, but with minCard=0 and a **reportText** Property value of **“**{No text}”**.**  In the following example, title value is empty, and the **altText** Property value contains the hidden Question text. This approach is used because in general, locked items are not intended to appear in the DEF or report. In rare cases, a locked element may be an essential piece of information that MUST display in the report. In these circumstances, the associated metadata will dictate the desired use and output. Form designers may also use custom Properties to embed required information into FDFs.



Example 41

## Flavors of Unanswerable (FOU)

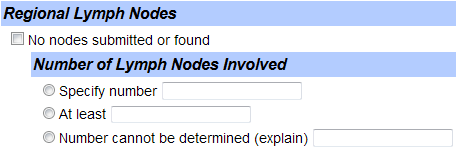
A special group of ListItems are useful when a Question cannot be definitively answered. Since these are similar to the well-known [HL7 “Flavors of Null,”](http://wiki.hl7.org/index.php?title=NullFlavor) they are called “Flavors of Unanswerable” (FOU). Some FOU ListItems are represented as an LIR, allowing the user to type in a more detailed explanation of why the Question cannot be answered. The LIR responses for these are almost always optional (responseRequired = “false”). A sample list of FOU ListItems follows:

* Cannot be assessed
* Cannot be determined
* Cannot be evaluated
* Indeterminate (this term, while still used rarely, has been [deprecated](#_Deprecated_Content_1) from clinical forms because its meaning has been interpreted inconsistently as either “Cannot be determined” or “Equivocal”)
* Inconclusive
* Not applicable
* Not identified (used to replace more assertive terms like “Not present”)
* Not known
* Not specified
* Unknown
* Unspecified

# Rules

Sometimes the response captured by one QAS determines whether other non-child QAS items or Sections should be activated, or the response is used in a calculation of data value(s). These types of cases can be handled by SDC *Rules*, which are mostly out of scope for this Primer. However, two such rules are currently in widespread use:

## Selection Disables Children

The selectionDisablesChildren **(SDAC[[56]](#footnote-57))** attribute generally applies to a LI on a Question with only one LI.[[57]](#footnote-58) Normally, the selection of a LI by the user will cause the first-level child controls to be activated. SDAC reverses this behavior – selection of the LI disables (deactivates) the child controls.



If the   
**LI-SDAC**control is selected, then this Question is deactivated

**LI-SDAC**

Example 42: Selection Disables Children (SDAC) XML

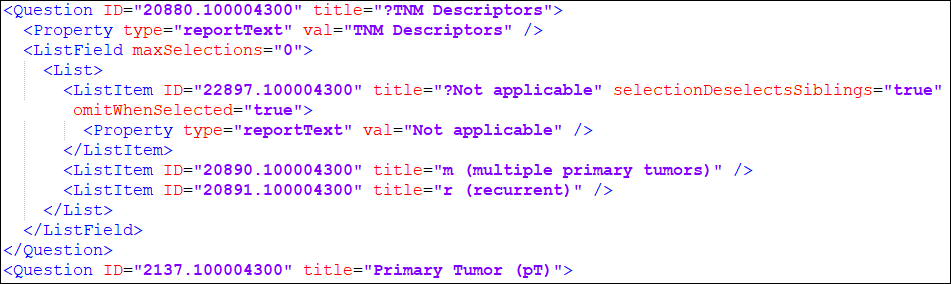
The reporting rules for SDAC items are essentially the same as those for reporting non-SDAC items. If an SDAC ListItem is unselected, the ListItem must not appear in the report. The parent Question of the unselected SDAC ListItem is also omitted from the report, because unanswered Questions are never reported. If the SDAC ListItem is selected, the report text of the SDAC ListItem and its parent Question’s title is controlled by the title value and **reportText** Property value, as usual. The activated descendant items of the *unselected* SDAC ListItem should appear in the report according to the usual reporting rules described for each item type.

Example 43: Selection Disables Children (SDAC) DEF

## Selection Deselects Siblings

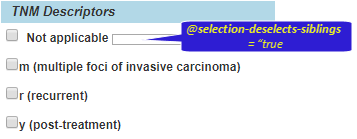
The selectionDeselectsSiblings **(SDS)** attribute applies to any LI with a parent QM. If SDC is set to true on an LI, then selecting the LI will cause the DEF to de-select all selected sibling LIs and ignore any user-entered data entered in the previously-selected sibling LIs and their descendant captured responses. Deselecting the LI (the one with SDS = “true”) permits the user to manually re-select the sibling LIs, and re-enable any descendant captured responses, if present.

Example 44: Selection Deselects Siblings (SDS) XML



If the **LI-SDS** control is selected, then any selected sibling LIs are deselected

**LI-SDS**

--

Example 45: Selection Deselects Siblings (SDS) DEF

## Question Rules

## ListItem Rules

## Events and Guards

### Form Events

### DisplayedItem Events and Guards

### Question Events and Guards

### ListItem Events and Guards

## Form-Level (Polling) Rules

### Validation Rules

#### IllegalSelectionTest

#### IllegalSelectionSets

#### ItemAlternatives

### Auto-Activation

### Auto-Selection

### Scripted Rules

### Call Rule

### External Rules (Web Services)

### ConditionalGroupActions

## Actions

### Action Types

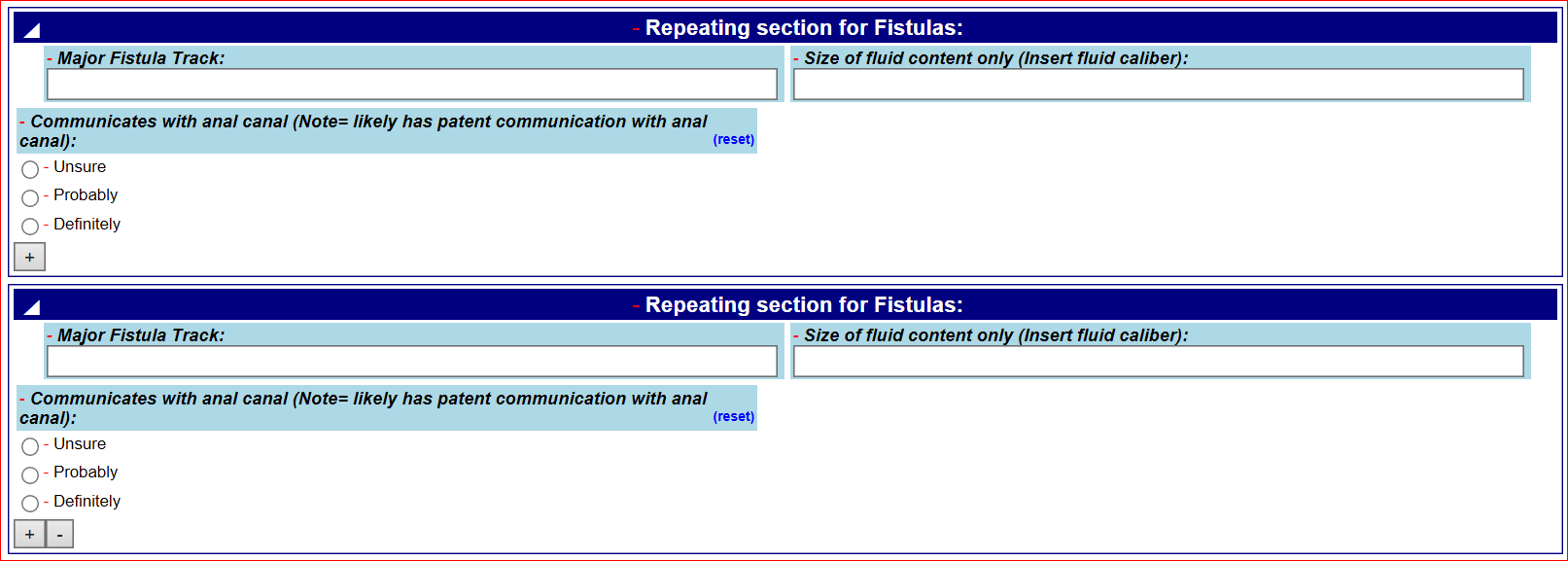
### ConditionalGroupActions

# ****Repeating Sections and Questions****

Section and Question components may be repeated any number of times in a DEF. The maxCard attribute specifies the maximum number of allowable repeats. If @maxCard=1 (the default) or is missing in the XML, the section or Question is not repeatable. If maxCard=0, then the number of repeats is unlimited.

If minCard=”0”, then the user is not required to capture a response(s) in the Section or Question. If minCard= “1” or greater, then the Section or Question must have its responses captured. If minCard >1, then the Section must be repeated for the number of times in the minCard attribute (this pattern is rare).

Only one instance of a repeatable Section and Question component should appear in a DEF when it is first rendered. The user must be able to indicate when to display the next repeatable Section or Question. The Form Filler must therefore include a clickable component (e.g., “+” and “-” buttons) or other interactive method with which the user can instruct the Form Filler to insert (or remove) a repeated instance of the Section or Question. This is illustrated in the following figure:



Add repeating **Section**

Delete repeating **Section**

Example 46: **Repeating** Sections **and** Questions

### Managing Repeated XFC and Component IDs with Monotonically Increasing Suffixes

When a Section or Question component is repeated, there must be a way to differentiate the IDs of each repeated DEF component. This is done by adding a suffix to the ID of each repeated component/XFC and its descendant components, as illustrated in the next XML figure.

For each repeated component, an integer suffix (represented by “#” below) is created and appended to each ID in the entire repeated component **block** (i.e., the repeated component and all descendants). To manage the manipulation of ID attributes, The FF/DEF contains an **integer counter** function. The integer counter # starts at 0, and is increased by one every time a new repeated component block is added by the user, anywhere in the XFC/DEF. The XFC suffix begins with two underscores (\_\_) followed by the shared counter integer (#) that identifies the repeated component block in the FDF. The \_\_# suffix is added to every XFC in the repeated block.

Example 47: Managing repeated XFC and component IDs with monotonically increasing suffixes



**Section** can be repeated up to 10 times

Repeated items: **@IDs** have “\_\_1” suffix

Original items

### Nested Repeats

As noted earlier, repeating blocks of XFCs are defined by the maxCard value, highlighted in cyan in the example below. Nested repeats that occur within a repeated block of XFCs (i.e., repeated blocks inside repeated blocks) are treated the same way as the top of the repeated XML block: the integer counter is incremented again by one, and the original XFC ID is followed by \_\_#, where # is the newly incremented integer shared by all IDs in the nested XFC block. The *original*, non-repeated XFC does *not* get a suffix on its ID value, but if it did, it would be “\_\_0”.

|  |
| --- |
| <Section ID="S1" title="Nodule:" minCard="0" **maxCard="10"**>  **<ChildItems>**  <Question name="nodNum" title="Nodule Number:" ID="Q1">  <ResponseField>  <Response>  <positiveInteger/>  </Response>  </ResponseField>  </Question>   <Question name="imgNum" title="Image Number:" ID="**Q2**" minCard="1" **maxCard="10"**>  <ResponseField>  <Response>  <string/>  </Response>  </ResponseField>  </Question>  **</ChildItems>**  </Section>   <Section ID="S1\_\_1" title="Nodule:" minCard="0">  **<ChildItems>**  <Question name="nodNum\_\_1" title="Nodule Number:" ID="Q1\_\_1">  <ResponseField>  <Response>  <positiveInteger/>  </Response>  </ResponseField>  </Question>   <Question name="imgNum\_\_1" title="Image Number:" ID="**Q2**\_\_1" minCard="1"   maxCard="10">  <ResponseField>  <Response>  <string/>  </Response>  </ResponseField>  </Question>  *Repeat* **Section:**  All **@ID** and **@name** values have a suffix of \_\_1 or \_\_2  *First* repeat of **Question Q1** uses same \_\_1 suffix on **@ID** and **@name**.  **Question Q1** cannot be repeated as a nested item (@maxCard is missing, and its default value is “1”)  *First* repeat of**Question** **Q2** uses same \_\_1 suffix on **@ID** and **@name**.  **Question Q2** can be repeated as a *nested* repeat item up to 10 times.  *First* repeat of Section S1 uses \_\_1 suffix  *Original* **Question** **Q2** can be repeated as a *nested* repeat item up to 10 times.  *Original* **Section** block; the entire block can be repeated up to 10 times.  No suffixes are used on **@ID** values.  <Question name="imgNum\_\_2" title="Image Number:" ID="**Q2**\_\_2" minCard="1">  <ResponseField>  <Response>  <string/>  </Response>  </ResponseField>  </Question>  *Nested* repeat of **Question Q2:**  **@ID** and **@name use** \_\_2 suffix.  *Second (Nested)* repeat of**Question** **Q2** uses same \_\_2 suffix on **@ID** and **@name**.  **</ChildItems>**  </Section> |

Example 48: Nested repeats

The first repeat of an XFC (the second instance of the XFC) gets a \_\_1 suffix on ID of all the XFCs in the repeated block. If a *nested* repeating XFC block (e.g., the Question with ID = “**Q2**”) repeats within the “\_\_1” nested block, then the counter is incremented, in this case, to “\_\_2”, and this new suffix is used for the internal nested block. In practice, if an XFC block already has an ID with a counter suffix, the counter is incremented for every ID in the additional repeated XFC block.

The optional name attribute is an identifier used to enable programmatic manipulation of captured data. If the name attribute is used, it receives the same suffix as the ID attribute.

# DEF Validation and Reporting Results

## Incomplete/Invalid DEF

A DEF is not considered complete (valid) unless responses to *all* applicable[[58]](#footnote-59) required[[59]](#footnote-60) Questions are captured. Prior to submission, the user must be notified which required item (i.e., any Question with minCard > 0, or any ListItemResponseField with responseRequired **= ”true”**) does not have a suitable response captured. Even if required responses are missing, the FF, in most cases, should allow the user to save the form.

## Validating, Saving and Reporting from the DEF

**DEF software must allow a user to save a DEF which has required Questions that are unanswered.**

Sometimes an attempt to save user responses in a DEF generates validation errors, e.g., missing responses to required Questions. Although an “invalid” DEF may be incomplete or contain errors, it reflects the real-world situation, in which some data may not be available to the end-user, or the DEF may be improperly designed.  Thus, SDC software must allow the end-user to save an incomplete template for later completion, review, editing and submission despite being incomplete.

DEF validation mechanisms should warn users when *applicable* required Questions are left unanswered or are answered in an illegal way (e.g., wrong data type). *Unanswered* refers to both missing selectable ListItems as well as required fill-in values (LIR and QR). Applicable Questions are those that can be reached in the QAS tree through the user’s selection of ListItems and entering appropriate responses in QR and LIR fields.

By default, unanswered/skipped Questions should not be reported, even if the Questions are required for accreditation. Local implementations may alter this default rule as they see fit.

## Validation and Reporting Test Template

# Generating Reports from a DEF

Reports contain the user responses from one or more DEFs. Reporting user responses is often the primary reason for using a DEF. The desired report output can greatly affect the design and usability of a DEF. Satisfying accreditation requirements may depend on report layout. For example, CAP requires that reports are laid out in [synoptic format](http://www.cap.org/ShowProperty?nodePath=/UCMCon/Contribution%20Folders/WebContent/pdf/cp-synoptic-report-definition-and-examples.pdf).

Using the DEF format as a guide to design reports can result in suboptimal report readability. However, the optimal design of reports for each use case is a controversial issue. Some general considerations for report design follow. Most are subject to personal preference.

* Reports generally do not contain unanswered Questions or unselected ListItems.
* In many cases, the Question or ListItem title text on the DEF should be altered to improve report readability.
  + As noted earlier, the DEF displayed text of each item is contained in the item’s title. The title text should also be used when generating reports from the DEF, unless the text is overridden by the value in the item’s reportText Property. If the title text should be completely suppressed in the report, then reportText Property will have a value of “{No text}”.
  + In some cases, changes to the title in the report can cause a misreading of the intended concept and can also cause an accreditation issue. Caution and testing is advised in all of these cases.
* The layout of a report may not always follow the hierarchical layout of the DEF. In some cases, a left-aligned format may be preferred to a hierarchical layout.
* The separation of Questions from ListItems can take several forms, e.g.,
  + Use of one column for Questions and another for ListItems
  + Separation of Question and ListItem by punctuation such as “:” or underscores or a string of dashes or dots.
* The display of multiple ListItems from a single multi-select Question can be accomplished with several formatting options. One popular format is to display comma-separated multiple ListItems consecutively on the same line. However, complexities arise, especially when some ListItems have fill-in values or sub-Questions. Other formats report each ListItem on a separate line.
* The inclusion of Sections and “Report Notes” in reports is optional. However, context, readability and important information may be lost if they are omitted. If a Section has no content (e.g., none of its subsumed Questions are reported), then, by default, the Section should not be reported.
* Some DisplayedItems are designed specifically for report output, and may not appear on the DEF.
* The ordering of Questions and ListItems in reports may not be the same as the DEF. However, it is generally acceptable to use the DEF ordering and the Section structure, if desired.
* The use of rich text (e.g., rtf or HTML), table formats, alternately-shaded lines, images and color can improve the readability and acceptability of reports for some readers.
* DEF responses may be reported in more than one location in a single report or in multiple reports. For example, a short diagnostic section may include output from a few QAS responses. The same QAS responses may also be present in a more verbose section of the same report or in a separate report.

A complete treatment of report formatting is out of scope for this document. Some additional report guidance and examples can be found in the [synoptic report](http://www.cap.org/ShowProperty?nodePath=/UCMCon/Contribution%20Folders/WebContent/pdf/cp-synoptic-report-definition-and-examples.pdf) examples available on the CAP website.

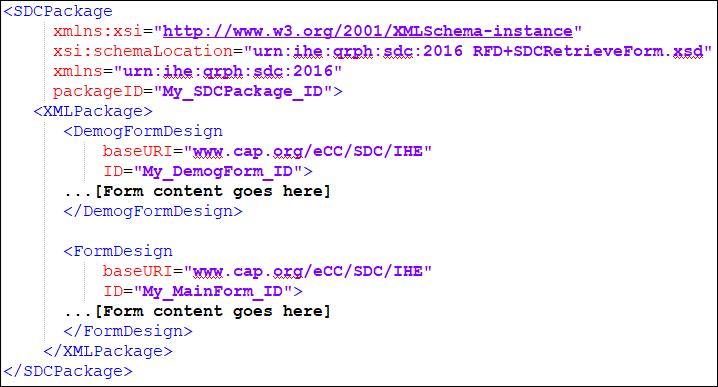
# SDC Instance Metadata

# The SDCPackage and its Metadata

## The SDC Package

Example 49: The SDC Package

FDFs are distributed in an XML wrapper called **SDCPackage**. An SDC Package can contain one or more FDFs along with metadata describing the contained FDFs and their usage. SDCPackage contains a child element called **XMLPackage**[[60]](#footnote-61), and XMLPackage contains the FDFs. The SDCPackage element contains **@packageID**, which uniquely identifies the set of packaged forms and related metadata in the SDCPackage. The FDF forms contained in the XMLPackage element (in the **DemogFormDesign** and **FormDesign** sub-elements) use the **@ID** attribute to identify each FDF form. The @ID attribute may be used in conjunction with the **@baseURI** attribute, [described later](#_The_@ID_and_1).



**SDCPackage** contains the following attributes:

* **xmlns:xsi** - This is the namespace declaration that allows the inclusion of XML Schema instance attributes in the SDC XML. The use of the “xsi” namespace prefix is only used to allow the inclusion of the xsi:schemaLocation attribute in the SDC XML.
* **xsi:schemaLocation** attribute has two values, separated by a space. The first value is the SDC namespace. The second value is the location of the XML Schema to use with that namespace.[[61]](#footnote-62)
* **xmlns= “urn:ihe:qrph:sdc:2016”** indicates the SDC namespace.
* **packageID** is the globally-unique ID of the SDC Package, which may contain more than one form.

The **XMLPackage** element, which is a sub-element of SDCPackage, also contains the **FormDesign** element, which holds the FDF content. XMLPackage may also contain optional FDF demographic content in another FDF element called **DemogFormDesign,** which if present, is placed before the FormDesign element in the XML layout. The FormDesign and DemogFormDesign each contain their own unique @ID attribute. DemogFormDesign is used to contain a generic demographic FDF that can be reused in combination with multiple different domain-specific FDFs. DemogFormDesign is separated out as a separate FDF so that its content does not need to be copied into and maintained with the many domain-specific SDC forms that need to use the same kind of demographic information. In this manner, demographic content need only be maintained in a single form suitable for one or more use cases in multiple FDFs.

## Package Attributes

## The Admin Element

## Package Nesting

## SubmissionRule

## ComplianceRule

## SDC HTML, XML and URL Options

### DemogFormDesign

## SDC Maps

# Pre-population of SDC Forms

## Form Pre-population (Pre-pop) and Auto-population (Auto-pop)

SDC provides for two ways to insert already captured EHR data to forms: pre-population and auto-population. A brief description of Pre-pop and Auto-pop follows. For more detailed information see <http://www.google.com/url?q=ftp://ftp.ihe.net/Quality/2014_2015_YR_8/QRPH_Technical/Technical%2520Supplements/SDC/archive/IHE_Suppl_Template_Rev1.0-SDC-20140415.docx&sa=U&ved=0ahUKEwjEmdLKg8rTAhWJZCYKHV8wCnsQFggoMAM&usg=AFQjCNGdCldlygydbKl2R-cVQO5VR_EViw>

## Pre-pop

The first of two ways to apply EHR data to a form is called pre-population. In this approach, an EHR exports a standard document, typically a templated CDA, to a FF that uses the data, in conjunction with an SDC map file, to populate fields in the form. Alternatively, the FF can query a data store, such as a FHIR resource, and populate the form with the returned data.

## Auto-pop

Auto-population is the population of an SDC form with data from the local EHR that has been embedded in the FDF. It enables a FF to apply those data directly to the form.

## Role and Structure of SDC Maps

# Submitting SDC Forms to FormReceivers

## Instance Metadata for SDC Package and Form Submissions

# The IHE SDC REST API

IHE RFD SOAP methods found at …

## Empty SDC Forms

## SDC Instance Forms

# Deprecated Content

The term “deprecated” is used in programming jargon to identify items that will soon be removed from use, and will then be unsupported. For the eCC templates, “deprecated” means that an eCC item has been “taken out of service” or removed from active use in a template. All eCC line items that were removed (“deprecated”) in an eCC template over the course of all previous versions (e.g., all versions of “Invasive Breast Cancer”), are accumulated under the “(Deprecated Items)” Section at the bottom of that FDF file. The deprecated items preserve the @IDs and metadata (e.g., the “required” attribute) that were previously released. However, the “sort-order” metadata for deprecated items reflect the new positions of these items in the XML files. Deprecated items should be ignored for implementation purposes. They are provided to aid backward-compatibility for queries across eCC versions.

## When are Items Deprecated?

Deprecation of Questions and ListItems can have adverse consequences on the use of queries that reference deprecated items. Therefore, items are deprecated only when the benefits clearly outweigh the implementation difficulties.

Items may be deprecated for several reasons. These include: improving clinical “correctness,” generally at the request of pathologists; improving support for NAACCR mapping; fixing of defective QAS modeling; and updating QAS to keep current with the latest CCP changes. Any item may be deprecated when it is dropped from the source CCP. Items may also be deprecated due to associated changes in the CCP, or changes due to eCC remodeling.

In general, Questions and ListItems are deprecated when keeping them is judged likely to adversely affect the semantics, context and/or integrity of database queries that analyze those items. Deprecation is avoided whenever the Question or ListItem can be reused without adversely affecting query and data integrity. Deprecation of DisplayedItems and Sections is unlikely to significantly affect query integrity, unless it affects the context in which QAS are presented. Thus, these items may be deprecated when the Section/DisplayedItem text semantics change significantly.

In the past, if semantics were not adversely affected, Sections, DisplayedItems and List Notes were occasionally changed into other types of items (including Questions and ListItems) without being deprecated. This practice has been changed, and deprecation in now used in these situations.

### Question deprecation occurs when:

* A single-select Question (QS) or multi-select Question (QM) changes to a fill-in Question (QR). This usually involves loss of ListItems.
* QS changes to QM.
* QM changes to QS
* QR changes to QS or QM. This usually involves addition of ListItems. This occurred multiple times with the new Margin measurement remodeling in this release.
* QS/QM/QR changes to DisplayedItem or Section.
* Question context changes how the Question is answered.
* Significant addition or deletion of ListItem choices, which is likely to significantly alter the selection frequency for the original set of ListItems.

### Question deprecation does not occur when the following isolated changes occur:

* Addition/deletion of some “less-significant” ListItems. These include ListItems like “None identified,” “Not applicable,” “Other (specify)” or “Other histologic type not listed above (specify)” etc. These are changes which would not generally affect the frequency at which the original ListItems would be selected.
* When an LI item changes to LIR, or LIR changes to LI, but the LI item is not itself deprecated (see below for the LI deprecation rules).

### ListItems are deprecated when the following isolated changes to the ListItem occur:

* For a given ListItem LI1, if more specific ListItems, which allow more discrete querying, are provided to replace or overlap with LI1, then LI1 should be deprecated.
* For a given ListItem LI1, if adding or dropping other specific ListItems in the ListItem list would create the likelihood that LI1 will be selected with altered frequency, then LI1 should be deprecated and replaced.
* A change in the ListItem text causes a change in the semantics
* The ListItem units change
* LIR changes to QR

NOTE: In the above cases, the Question should also be deprecated

### ListItem deprecation generally does not occur when the following isolated changes occur:

* If a ListItem (LI) changes to LIR, but the Response part is only a *general comment* (free text) field.
  + However, if the new Response part is a *specific response* to a sub-Question contained in the ListItem’s **title** text, such as “Specify type”, or “Other histologic type”, then the semantics of the ListItem have changed, and the LI must be deprecated and replaced with an LIR.
* If LIR changes to LI, when the deleted Response part is only a *general comment* field, and not a *specific response*, as defined above.
  + If the removed Response part is a *specific response* to a sub-Question contained in the ListItem’s **title** text, then the original LIR must be deprecated and replaced with an LI.
* If the Response metadata (e.g., **@responseRequired**) of an LIR is changed, and
  + The Response part is only a general comment (free text) field,

--OR--

* + The metadata change does not affect the LIR semantics or interpretation. For example, a change in units (e.g., cm to mm) would require deprecation of the LIR, but a change of **@datatype** from decimal to integer would not require deprecation in most cases.
* Parent QS changes to QM
* Parent QM changes to QS

# Versioning

## FDF Versioning

## Package Versioning

## eCC Versioning

Each FDF filename begins with a shortened and concatenated form of the actual template name, followed by a version identifier, followed by the “.xml” suffix (e.g., “BreastCompleteExcisionand\_2.000.011.xml”). The *FDF version (00*2.000.011*)* is incremented to mark any changes to the XML file since the previous release. Please note that the *FDF version identifier* is also present in the header section of each FDF file, using the attribute name “template-xml-version”.

The incrementing of template version identifier follows a defined pattern, as described below:

### The eCC FDF Version:

Each FDF filename begins with a shortened and concatenated form of the actual template name, followed by an FDF version identifier (FDF-VID), and then followed by the “.xml” suffix (e.g., “BreastCompleteExcisionand\_2.000.011.xml”). The FDF-VID (002.000.011) is incremented to mark any changes to the XML file since the previous release. The FDF-VID is also present in the header section of each FDF file, using the attribute name **@ID**.

In some cases, the XML files may change, even though the eCC data is unchanged. This may occur, for example, when the XML schema changes, or when new eCC metadata (e.g., the “data type”) has been released as new XML attributes in an XML file.

The full format of the Version ID is 123.456.789.1000043.

* Section 1 (“123”) is incremented only for major changes in a CAP Cancer Protocol (CCP), such as a complete rewrite, or a major change in AJCC version.
* Section 2 (“456”) is used for changes that alter the Item [@IDs](#_The_Ckey_Format:) in an active eCC template.

Item @ID changes occur when Questions or ListItems are added or retired (“[deprecated](#_Deprecated_Content_1)”) in a template. This also occurs when the semantics (i.e., the meaning or context) change in a Question or ListItem. Examples include significant rewording of a Question or ListItem, addition of new line items (e.g., adding a ListItem for “Not Applicable”), addition or deprecation of QAS, and deprecation (or “un-deprecation”) of individual Questions or ListItems. In rare cases, a change in Sections, DisplayedItems or instructions can alter the way that a Question is answered, thereby changing the context and/or semantics of Questions and ListItems. This type of change causes ambiguity in the meaning of the template items over time, requiring deprecation of the affected template items (and their @IDs), in conjunction with the creation of new Questions and/or ListItems (with new @IDs).

* Section 3 (“789”) is used for minor changes to the eCC template content in an active eCC template. The first 2 digits (“78”) are used to indicate the version of the minor content change (e.g., spelling errors, changes to the visible text that preserve item semantics, or changes to the position of items) or metadata change (e.g., a changed data type that does not alter the fill-in semantics). The last digit (“9”) is used to indicate that the XML release file has changed. This digit may be incremented even when there have been no content changes, e.g., when the XML schema has changed. In this case, all the previous digits (“123.456.78”) will remain the same as the previous FDF-VID, and only the last digit will be incremented. Released content changes always force an FDF change. Therefore, the last digit (“9”) is always set to “1” whenever there is a content change.

The namespace portion of the Version (“1000043”) is often omitted in the FDF file name for brevity, since all CAP releases use this namespace. Leading and trailing zeros may also be omitted for brevity.

#### eCC FDF Versioning Examples:

* 00**2**.000.00**0** original version 2
* 002.000.00**1** xml change only (e.g., new attribute) [new xml]
* 002.000.00**2** another xml-only change (e.g., minor or major schema change) [new xml]
* 002.000.0**11** minor new or changed content [+ new xml]
* 002.000.0**21** minor new or changed content [+ new xml]
* 002.00**1**.00**1** QAS structures modified – new @IDs, [deprecated](#_Deprecated_Content_1)/retired @ID [+ new xml]
* *Changed 10-31-2011:* 002.00**2**.0**01** moderate (@ID changes) and minor (e.g., fixed typos) changed content [+ new xml] – Any higher-level change will reset the lower level changes to 0. However, the XML digit is always reset to “.001”
* 00**3**.000.00**0** completely new version, sometimes with all new @IDs (e.g., new version based on heavily revised CAP Protocol document). A major change in a staging system (e.g., the updates in the AJCC 8th edition staging system) will usually result in a version increment in section 1 of the version ID.

### Relationship of FDF Versions to CCP Versions

Introduced in Feb 2011, the new versioning ID system for the CCPs is not intended to cover the same types of versioning issues as the eCC versioning system for FDF files. CCPs are versioned for content changes, whereas eCC FDF files may be versioned for content changes and/or technical changes. Because CCP version changes may or may not affect the case summary section, CCP version updates may or may not affect a later FDF release version. Thus, any given version of an FDF file could remain constant through several versions of the parent CCP document. Conversely, an FDF file may undergo several version changes while the parent CCP remains in the same version.

In general, information flows from the CCP to the eCC. In some cases, however, eCC-specific version updates (e.g., PERT-approved modifications that were added to support cancer registries) may be adopted in future versions of the CCP, if approved by the protocol authors and the CAP Cancer Committee.

# SDC Security Considerations

# The Basic XML Structure of FDF Files (old)

When reading this section, it will be helpful to open any of the eCC FDF files in an XML colorizing tool such as Oxygen, XML Spy or Notepad++. Reference to the enhanced eCC Schema file and documentation will also be helpful.

Refer to the FDF file content fort examples when reading the descriptions below.

FDF files consist of several main sections:

* **XML declarations**, including a reference to an XSL transform style sheet used for rendering templates into HTML
* The **<template>** root element
  + The **<required>** element: Set to “true” for required templates, “false” for optional templates.
  + The **<template-header>** element: This section contains metadata about each template version, as well as a minimal set of display data for rendering the FDF file as an HTML document.
  + The <**template-body>** element: This section contains the line items (e.g., Sections, Questions, ListItems, and DisplayedItems) that make up the content of each template version.

The elements **<template-header>** and **<Section>** can be confused**.**  Note that **<template-header>** occurs in the FDF’s **<template>** section, and **<Section>**occurs in the <**template-body>** section.

## Structure of FDF Elements and Attributes for Template Line Items:

The content of each eCC XML file reflects the CAP Cancer Case Summary items very closely. From the case summaries, we identified a small number of recurring types of line items for XML representation in FDF files. In the explanations below, note that lists of ListItems are always embedded within a **<Question>** element and a **<fixed-list-answer>** element.

* **Sections** (H)  
  **<Section>**
* **Notes** (N)  
  **<note>**
* **Single-Select Questions** (QS)  
  <**Question**…>… <**fixed-list-answer** allow-multiple-selection="false">
  + Selectable ListItem (e.g., combo box, list box, or set of option buttons) (A)  
    **<ListItem>**
  + Selectable ListItems that also request a response if that ListItem is selected (LIR)  
    **<fixed-list-fillin-answer>**
  + List Notes, which appear interspersed with ListItems (LN)  
    **<DisplayedItem>**
* **Multi-Select Questions** (QM)  
  <**Question**…>… <**fixed-list-answer** allow-multiple-selection="true">
  + ListItems (e.g., checkboxes, multi-select list box) (A)   
    **<ListItem>**
  + ListItems that also request a fill-in response if that ListItem is selected (LIR)  
    **<fixed-list-fillin-answer>**
  + List Notes, which appear interspersed with ListItems (LN)  
    **<DisplayedItem>**
* **Response Questions** (QR) – there is no list of ListItems for QR  
  <**Question** …. Question-fillin="true">

## Metadata in <template-header >

### Identifiers

The following identifier attributes are found in the **<template-header>** element at the top of every FDF file:

* **@template-id**: a unique @ID-formatted identifier that is shared by all the line items in a specific version of a template. For example, all versions of the Invasive Breast Cancer template share the same **@template-id**. If major content changes are made to the template (e.g., a template is split into 2 templates or 2 templates are grouped into a single template), a new template-id may be assigned, and all line items are recreated with new line-item @IDs. In this case, the retired template(s) are [deprecated](#_Deprecated_Content_1), along with all of the items contained in those templates.  
    
  Note that the **@template-id** identifier resides in the realm of template identifiers; it is not an identifier for line-items (e.g., Questions, ListItems, etc.). Therefore, a **@template-id** may share the same numeric value as a line-item identifier, but they represent different classes of data.
* **@checklist-id**: describes the template “type” for all versions of a template, as defined by the CAP Cancer Committee. For example, all Invasive Breast Cancer templates share a common **@checklist-id** representing “Breast Resection,” even though there may be more than 1 version of this template with separate **@template-id** values.
* **@template-xml-version**: This is the FDF version identifier (FDF-VID) described earlier; the version value is incremented whenever a template’s FDF file changes, even if the template’s semantic content and metadata are unchanged.

Note that each template version has a different **@template-id**, and each **@template-id** may associated with more than one FDF file version, each with a unique **@template-xml-version** attribute. The above one🡪many patterns may be expressed as **@checklist-id** 🡪 **@template-id** 🡪 **@template-xml-version**.

### CCP Metadata in the FDF file

* **@cap-cancer-protocol-name**: Name of the CCP that contains the case summary source of the FDF.
* **@cap-cancer-protocol-version**: The version identifier for the each CCP. This identifier was introduced with the Feb 2011 release, and will be updated whenever changes to the CCP are posted to the [CAP web site](http://www.cap.org/cancerprotocols). A document describing the CCP versioning system is included with the eCC release documentation. A similar document is available on the [CAP web site](http://www.cap.org/apps/docs/committees/cancer/cancer_protocols/workaid/Version_Code_Description.doc).

### Other Metadata

The following elements and attributes are contained in the **<template-header>** element:

* **<title>** element: The visible text for the template name, as found in the CCP
* **<category>** element: A grouper for displaying the templates used on the CAP web site
* **<versions>** element: A grouper for versioned classification systems (e.g., “AJCC-UICC”) linked to the current template. Each related versioning system includes the following attributes in the <**version>** element:
  + **@display-name**
  + **@major-version**
  + **@minor-version**
* **<publication>** element: A grouper for:
  + **<web-posting-date>** element: The date the CCP was posted to the CAP web site
  + **<approval-status>** element: The value will always be 1, or missing for released FDFs.
* **<generic-header>** element: The running header on the CAP protocol Word/PDF documents, e.g., “Surgical Pathology Cancer Case Summary”
* **<restrictions>** element: Short instructions describing when a template should (or should not) be used

## Metadata in <template-body>

### Identifiers

Each line item in a template version receives a unique key value in the @ID format. For historical reasons, this unique set of @ID values had been split into 4 different named identifiers, although these 4 identifiers derive from the same set of unique, mutually-exclusive keys. Thus, 4 identifier attributes for template line items are found embedded in each FDF <**template-body>** element:

* **@Section-id**
* **@Question-id**
* **@answer-id**
* **@note-id**

To reiterate, all identifiers in the identifier group, despite bearing 4 different names, belong to the same unique key space for *line item @IDs*, such that no line item @ID is ever shared with another line item. In other words, each line-item @ID value uniquely identifies a line item in a template, regardless of the item type. When the generic term “@ID” is used, it usually refers to one or more of the above 4 line-item ID identifiers.

When minor template changes are released with new [FDF versions](#_Versioning_of_XDT_2), the same line item identifier attribute values continue to be used in the new version of the FDF file. As noted earlier, these identifiers are only changed ([deprecated](#_Deprecated_Content_1) or added) when CAP template changes would disrupt the context, semantics and/or structural integrity (i.e., the modeling style) of the template.

The following sections describe the elements and attributes found in the <**template-body>** element of each FDF file. The color-coding of template item types (e.g., Question) should enable the reader to rapidly locate the important XML elements and attributes for a given item type. For example, scanning the page for all Question items will rapidly locate the elements and attributes that apply to each Question item.

### Generic Metadata:

* **<title>** element:
  + May appear on element:Section, DisplayedItem, ListItem, ListItem, and Question
  + Default value: “”
  + Function:
    - Visible text for Sections, Questions, ListItems, and DisplayedItems
* **<text>** element:
  + May appear on element: DisplayedItem
  + Default value: “”
  + Function:
    - Visible text for DisplayedItems
* **@sort-order**:
  + May appear on element: DisplayedItem, Section, DisplayedItem, ListItem, ListItem, and Question
  + Default value: null
  + Function:
    - Determine the correct display order of all template line items in and FDF file
    - Sort order is a convenience for processing the XML QAS tree in sequence, especially when QAS ListItem sets are stored in a database and need to be sorted into their original FDF-occurring order.
* **required** element ([*deprecated*](#_Deprecated_Content_1) *in enh XML, but still in use*):
  + May appear on element: **template**, Section, Question
  + Default value: null
  + Function:
    - set to “true” when the template or item is required for accreditation

### Question Metadata:

* **@alt-text:**
  + May appear on element: Question
  + Default value: “”
  + Function:
    - contains suggested text for Questions when the **<title>** element for the Question is blank. See the section “[Invisible Question Text](#_Invisible_Question_Text)” for more information.
* **@allow-multiple-selection:**
  + May appear on element: Question\fixed-list-answer
  + Default value: false
  + Function:
    - when the value is “true”, this attribute indicated that a Question is multi-select (e.g., implemented with one or more checkboxes as ListItems, QM). If the attribute is missing or “false,” then the Question element holds single-select ListItems (e.g., implemented with a combo box or option buttons)
* **@readOnly**:
  + May appear on element: Section, ListItem, ListItem, and Question;
  + Default value: false
  + Function:
    - When set to “true,” supports *pre-answered* (i.e., default) Question/ListItem data that should be included when storing template data. For example, there are rare situations where there is only one possible specimen type for a template. This Question/ListItem pair is locked (non-editable, read-only), and is intended for transmission as part of the ListItem set when a template is filled-out. If **@readOnly** = “true” on a S, N or Q, then all descendants (e.g., ListItems) are also locked. If **@readOnly** = “true” on an LI, only the LI is affected, but not the descendants (if present). All ListItems subsumed under a locked S or Q are treated as selected. All locked ListItems are treated as selected.

### QR and LIR Metadata:

* **@datatype:**
  + May appear on element: ListItem, and Question
  + Default value: “”
  + Function:
    - Indicates whether a fill-in text field accepts string, integer or decimal values. This value list may be modified or expanded in future releases.
* **ResponseUnits:**
  + May appear on element: ListItem, and Question
  + Default value: “”
  + Function:
    - provided to guide the user in data entry and for aid in producing reports from eCC data. Values include mm, mm^2, cm, grams, %, mitotic figures.

### ListItem Metadata

* **@selectionDisablesChildren** (SDAC)
  + May appear on element: ListItem, ListItem
  + Default value: false
  + Function:
    - If SDAC is true and the LI or LIR is selected, then all descendants of the A become disabled
* **@selectionDeselectsSiblings** (SDS)
  + May appear on element: ListItem, ListItem
  + Default value: false
  + Function:
    - If SDS is true and the LI or LIR is selected, then all selected sibling ListItems are deselected

## Deprecated Metadata:

The following undocumented elements and attributes have been *deprecated* from the FDF files since the April 2010 release:

* **@ID** attribute: found in FDF **<template>** element; replaced by **<checklist-id>** in the FDF header
* **@template-version** attribute: found in FDF **<template>** element; never used
* **@versionID** attribute: found in FDF **<template>** element; duplicated the **<template-id>**
* **<length>** element: found in FDF **<template-body>** element; not possible to use without additional validation metadata

Deprecated in 2016 for eCC enhanced XML:

* <authority-required> element, found under Question element
* <required> (but still in use)

# New and Changed Content in eCC enhanced XML (old)

The original (“legacy”) eCC XML format, described above, was unable to express metadata for features such as report output, CR behavior and repeating elements. The eCC enhanced XML (enh XML) format is designed to support these new features, while serving as a bridge to the newer Structured Data Capture (SDC) model. The enh XML was first previewed in the March 2016 CAP eCC release. It was developed to be a drop-in, backwards-compatible replacement for the original (“legacy”) eCC format. The basic structural features of the eCC XML are unchanged. Except for the deprecation of an unused element, all the changes take the form of adding new attributes to existing eCC XML elements.

CR and required item issues were a special focus in the development of the enh XML. In legacy versions of the eCC XML, optional status was handled simply with the **<required>** element. This element could take a value of true or false. The legacy element “**<authority-required>**” had an essentially identical function to **<required>**, and therefore **<authority-required>** was removed entirely from the newer eCC enh XML. However, a single element for required status cannot completely convey the nuances of handling CR Questions in a DEF. To address the distinction between required, CR and optional items in the eCC XML, four new enh XML attributes are used together: **@mustImplement**, **@minCard**, **@responseRequired** and **@omitWhenSelected**.

To better support text output that is tailored for reports, new reporting attributes include: **@reportText** and **@textAfterResponse** .

To support computer-readable repeating sections, **@minCard** and **@maxCard** were added.

The following section is a summary of the enh XML changes that are currently implemented. For each new attribute, the parent XML elements are color-coded. By scanning similarly colored patterns (e.g., Question) the reader should be able to rapidly locate all new attributes for a given element type.

## Deprecated XML element

The *authority-required* element (and its content) for Questions has been [deprecated](#_Deprecated_Content_1).

## New XML attributes

1. **@filename:**

* Appears on element: **template**
* Default value: “”
* Function:
  + Contains the original name (name in release folder) of the current file. Enhanced eCC file names end with “\_enh.xml”.
  + If the file name is changed on disk by a user, this attribute can be used to track the original file name.

1. **@mustImplement**:

* May appear on element: DisplayedItem, Section, DisplayedItem, ListItem, ListItem, and Question
* Default value: true
* Function:
  + When true or missing: the data-entry form must display the XML element.
  + When false: display of the element (including all descendants) is optional, but if it is displayed, then it must be answered when **@minCard** > 0 (see **@minCard** below).

1. **@minCard**:

* May appear on element: Section and Question
* Default value: 1
* Function:
  + Indicates the minimum number of repetitions for Section and Question.
  + When 0, indicates that accrediting agencies consider the Question to be optional to answer.
  + When 1, indicates that accrediting agencies consider the Question, when reached by a user in the form hierarchy, to be required to answer.
  + When >1, the Question is required to answer and should be repeated at least the number of times in the value of minCard.
  + If **@mustImplement** = “false” and **@minCard** > 0, then the Question is optional, but if it is displayed and answered, it should still obey the minCard requirement.
  + If **@mustImplement** = “true” and **@minCard** = “0”, then the Question must be displayed, but an answer is not required.

1. **@maxCard**:

* May appear on element: Section and Question
* Default value: 1, indicating that the element cannot repeat
* Function:
  + When 1, indicates that the element cannot repeat.
  + When > 1, indicates that the element can repeat a number of times up to the value of **@maxCard**.
  + When 0, indicates that the Question may repeat an unlimited number of times.

1. **@reportText**:

* May appear on element: DisplayedItem, Section, DisplayedItem, ListItem, ListItem, and Question
* Default value: empty (“”), indicating that the element **<title>** should be output in the report with no changes.
* Function:
  + If **@reportText** is empty (“”) or missing, then the **<title>** text is used verbatim in the report (assuming that other attributes do not prevent the reporting of this item).
  + If not empty (“”) or missing, **@reportText** contains the text that should appear in the report, instead of the **<title>** text.
  + If the **@reportText** value is “{no text}”, then no text should be output for the element. This is applied to most DisplayedItems, and many Questions.

1. **@responseRequired**:

* May appear on element: ListItem
* Default value: false, indicating that the user does not need to enter a fill-in value.
* Function:
  + When true: user must fill out the data-entry (fill-in) section of the list item.
  + When false or missing: filling out the data-entry (fill-in) section is optional.
  + All user fill-in responses must be displayed on the report, even if **@responseRequired** is false. The fill-in response is usually separated from the **<title>** by punctuation such as a comma or colon, unless the **@reportText** is set to "{no text}".

1. **@textAfterResponse**:

* May appear on element: ListItem and Question (QR only)
* Default value: “”
* Function:
  + **@textAfterResponse** is fixed text that appears after (to the right of) the user's response on the data entry form and report. This may be text for units such as "Millimeters (mm)", **"Centimeters (cm)",** etc. This text should also appear on report output.

1. **@omitWhenSelected**:

* May appear on: ListItem and ListItem
* Default value: false
* Function:
  + When true: the entire Question, all ListItems, and all descendants should be omitted from the report.
  + When false or missing: the Question and selected ListItem(s) should be present in the report, if the ListItem is selected. However, if the Question was not answered for any reason, it should be omitted from the report.

## Enhanced XML Usage Notes:

The following attribute values toggle "off" the state of all descendant items as well; If the parent item is not answered or displayed, then neither are any of the descendants:

**@minCard** = "0"

**@mustImplement** = "false"

For example, if a Question has minCard = “0”, and it is not answered, then none of its descendants may be answered. However, if the item later becomes answered/displayed/reported/visible due to local or user preference or due to user interaction with the form, then the child attributes take effect normally. For example, if a user chooses to answer the "optional" Question with **@minCard** = "0", then any child items behave according to the attributes that were set on them.

1. As noted above, all user fill-in responses must be displayed on the report, even if **@responseRequired** is false. An LIR fill-in response is usually separated from the Question **<title>** value by punctuation such as a comma or colon, unless the Question’s **@reportText** is set to "{no text}". In the latter case, report output must be customized and validated.
2. If **@textAfterResponse** has a value, it should be included in the report, unless a careful review of the report output determines that it is not needed or redundant.
3. **@readOnly**: when readOnly = "true" on a Question and a ListItem, the Question/ListItem pair should be included in the saved response set and transmitted with the report data to recipients. The locked data is designed for use by analysts querying the data sets. However, in almost all cases, the locked items should not be displayed on the data entry form or on reports. Therefore, in general, **@mustImplement** = “true”, **@visible** = "false" and **@reportText** = "{no text}" on locked items.
4. The format of report output cannot be completely specified by currently-available enh XML metadata. All report implementations should be validated by DEF and report end-users.

The table below contains a high-level summary of new or improved enh XML functionality.

## Comparison of eCC Legacy and enh XML

|  |  |  |  |
| --- | --- | --- | --- |
| **Functionality** | **Legacy XML method** | **enh XML method** | **Comments** |
| **Conditional Reporting - Questions** | ? on Question  Required = "true" or "false" | **@**mustImplement = "true" (default)  minCard = “0” | "?" will remain in **<title>** text for now; In the future, the "?" symbol may be removed. |
| **Conditional Reporting - ListItems** | ? on answer | **@**omitWhenSelected = "true" | "?" will remain in **<title>** text for now; In the future, the "?" symbol may be removed. |
| **Optional**  **Questions and ListItems** | + | **@**mustImplement = "false" | When **@mustImplement** = "false, "+" will appear in [eCC HTML text](#_eCC_Reference_Implementation); In the future, the "+" symbol may be removed. |
| **Report Text** | Remove special text | **@**reportText | Specify report text, when the report output test must be different form the data-entry form display. |
| **Text after Response** | none | **@**textAfterResponse |  |
| **Report Note** | Search for special text | **<text>** = "false";  **@**reportText = *value* | Specify that certain notes (“Report Notes”) to appear in reports, even if they do not appear in the DEF |
| **Repeats** | Search for special text | **@**minCard, **@**maxCard | Replaces instructional notes and text to handle repeats. |
| **Required** | required | **@**required is retained for backwards compatibility only | No change to required, However, minCard will replace required in SDC |
| **Authority Required** | Present, not used | Removed from XML | Obsolete |
| **Required Response for LIRs** | Search for special text | **@**responseRequired = “true” | Easier to discern when fill-in is required; no need to parse **<title>** of ListItem |
| **Omit from Report** | Not supported | **@**reportText = ”{no text}” | Easier to implement |

# eCC Implementation Using Composite Keys (@IDs) for eCC Template Items



## Introduction

Each versioned eCC template can be viewed as a specification (template) for one computerized Data Entry Form (DEF). Each template is specific for a particular version. For example, a Colon Cancer template that exists in 2 versions would be represented by 2 separate templates; each template would generate 2 separate, but related, DEFs.  For the eCC, each template is distributed as a separate XML document, called an XML Document Template (FDF). Each FDF has a version identifier called an “FDF Version ID” (FDF-VID). Please see “CAP eCC FDF Versioning” for more information on FDF versioning.

## @IDs and Templates:

The FDF content defines the **template** **items** (e.g., Questions, ListItems, Sections and DisplayedItems) that represent the lines of the paper case summary, and specifies the content and general layout of the computerized DEF. The XML for each template item is also able to represent metadata (e.g., parent-child structure, required status, etc.) associated with the particular template item.

To create an unambiguous system of referring to each item in each FDF, template items are assigned a unique line-item identifier. These identifiers (@IDs) are generated at CAP by using a unique integer value.

However, it is possible for end-users to customize FDF documents, and assign their own integer values to their custom template items. If multiple centers (including the CAP) assign simple integer values, inevitably some centers will end up using the same integer keys for different customized template items. Since these keys may be used to communicate data between centers, the use of simple integer keys would result in confused and corrupted data.

Therefore, the eCC metamodel employs a common solution for this problem: We append a site-specific “namespace” to the base integer of the @ID. ~~The CAP maintains a namespace registry, and will assign a unique ID number to any institution that requests one.~~ For each end-user institution, its unique namespace integer is appended, after a decimal point, to the base integer @ID, as follows:

[Local Unique Integer] + decimal point + [CAP-Assigned Unique Namespace]

123456 + “.” + 1000043 = 123456.1000043

In general, @IDs for template items are identified in the FDF files by the suffix of “-id.” Although @IDs for **Sections** and **DisplayedItems** are also provided in the FDF, there is no need to export or store those values as part of the ListItem set (see below for a discussion of data storage).

## Data Storage, Terminologies, and Transmission

The basic building block of FDF files is the Question-Answer Set (QAS). The QAS model follows the common practice of representing information as Question-answer pairs. In the eCC world, this translates into storing a Question @ID and a ListItem @ID for each Question and ListItem in the filled-out DEF. Although it is possible to use SNOMED CT, LOINC or other semantic coding systems for this purpose, complete sets of these codes for each FDF are not always available or stable over time. Therefore, the @ID is a convenient surrogate key for the Questions and ListItems. @IDs are mapped, in separate XML mapping files, to the appropriate semantic codes, depending on code availability and end-user needs. Thus, @IDs provide unique identifiers for lines in a template that can map to external coding systems such as LOINC, SNOMED CT, and NAACCR data items.

To enable querying based on external coding systems, code tables for these systems should be stored separately from the @ID-based QAS data. External codes should be linked to the Question/ListItem pairs by using the @ID as a “foreign key” in a mapping table for the external codes. The eCC team provides these @ID/external code mapping tables in XML format, so there is no need to maintain them on your own.

Since many coding systems (e.g., SNOMED CT and ICD-O-3) introduce changes to the code values or the meaning of codes, storing external codes together with the @IDs would require regular data maintenance, would risk the introduction of errors, and might result in the editing of an electronically-signed and locked patient record. In addition, external codes may not be available at the time of data storage. By using a mapping table, external codes can be simply referenced via the @ID; updates to the mapping table can be as simple as importing an updated eCC mapping file.

For some applications, it may be necessary to transmit semantic (e.g., SNOMED CT, ICD-O-3), terminological (e.g., LOINC) or use-case-specific (e.g., NAACCR codes) codes *in addition* to @IDs. Creating HL7 messages that include non-@ID coding systems is straightforward when using the provided eCC mappings between @IDs and external coding systems. In Canada, the exact types and locations of the codes in HL7 messages are subject to standardization by the North American Association of Cancer Registries (NAACCR), and also by governmental agencies such as Canada Health Infoway (CHI), Canadian Partnership against Cancer (CPAC) and Cancer Care Ontario (CCO).

### Terminology (ICD-O-3 & SNOMED CT) Maps

CCPs and eCC FDF files are released before the ICD-O-3 and SNOMED CT encodings are available. In general, limited @ID to terminology maps will be provided 6-12 months after FDF files are released.

To provide a functional and interoperable implementation of the CAP eCC, it is critical that neither SNOMED CT codes (i.e., Concept IDs), nor codes from any other terminology, are used as *unique identifiers* for CAP eCC Questions and ListItems.  @ID values should be stored for each Question and for each selected ListItem in the template.

### Data Storage

When storing the ListItem data for a Single-Select Question, a single Question code (the “@ID” of the Question item) and a single ListItem code (the “@ID” of the ListItem) are stored in the database.

When storing ListItem data for a Multi-Select Question, a single Question code (the @ID of the Question item) and a single ListItem code (the @ID of the ListItem) are stored in a database record for each selected ListItem (checked box).

Why is it important to store the Question @ID in addition to the ListItem @ID? During eCC updates, sometimes Questions are [deprecated](#_Deprecated_Content_1) to indicate a change to the Question’s value (ListItem) list, but some or all of the original ListItem remain in the list (they are not deprecated). Replacement of the Question (and its @ID) ensures that new queries based on the previous Question @ID will not return ListItems from the new Question. This is intended to prevent misinterpretation of query data when values to the value list (ListItems) change. In other words, queries based on the deprecated Question will not return combined results from the new and the deprecated Questions, since these Questions are based on different ListItem sets.

If the Question has not been answered, then neither the Question @ID nor the ListItem @ID need be stored. In the case of LIR items, the text of the ListItem’s Response section should be stored in addition to the ListItem’s @ID.  Following this model will help to ensure that the stored data is interoperable with other template implementations.

### Usage Examples

Following this model will help to ensure that the stored data is interoperable with other template implementations. (Please note that the examples below do not include local primary keys, other record/row identifiers, patient identifiers, terminologies or audit fields, etc., as these may vary between implementations.)

#### Single-Select Items (e.g., combo boxes, option buttons and lists):

When storing or transmitting ListItem data for a Single-Select Question (QS), a single Question code (the “@ID” of the Question item) and a single ListItem code (the “@ID” of the ListItem) should be stored or transmitted.  For example, if the user selects the ListItem “A1” from the QS Question “Q1,” the data may be stored as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Question @ID | ListItem @ID | | Response\* | Repetition Index\* |
| [Q1 @ID] | | [LI1 @ID] | NULL | NULL |

\* To be described below

#### Multi-Select Items (e.g., checkboxes and lists):

When storing or transmitting multiple ListItems for a Multi-Select Question (QM), each ListItem @ID is used together with the associated Question @ID. The Question @ID thus repeats for each selected ListItem. For example, for a Question (Q1) with selected (checked) ListItem A1, A3, and A10 (out of, say, 15 possible checkbox ListItems), the following pattern should be used:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Question @ID | ListItem @ID | | Response\* | Repetition Index\* |
| [Q1 @ID] | | [LI1 @ID] | NULL | NULL |
| [Q1 @ID] | | [LI3 @ID] | NULL | NULL |
| [Q1 @ID] | | [LI10 @ID ] | NULL | NULL |

\* To be described below

#### Response Questions (which have no ListItems):

In the case of Question Response items, the Question @ID should be stored as above, but the ListItem @ID should be left blank. In addition, the text of the Filled-in data should be stored or transmitted in a text field.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Question @ID | ListItem @ID | | Response | Repetition Index\* |
| [Q1 @ID] | | NULL | [Response text or number] | NULL |

\* To be described below

#### LIRs (Single- or Multi-Select LIs that allow text to be entered after the LI is selected):

In some cases, the selection of a specific LI requires or allows that a Response section (e.g., text and numeric) be completed.  This is known as a “ListItem-Response (LIR).”  LIRs will require that both the LI item key (“@ID”) *and* the Response text value be stored in the database, usually as part of the same database record, as shown here:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Question @ID | ListItem @ID | | Response | Repetition Index\* |
| [Q1 @ID] | | [LI1 @ID] | [Response text or number] | NULL |

\* To be described below

#### Repeating Sections:

Occasionally, single ListItems, a QAS, or a block of multiple QAS sections may need to be repeated on the DEF and in the database. The number of times that a Section or Question may repeat is determined by **@**maxCard, described earlier. For storing eCC data from repeating items, the implementation should support the ability of @ID-based entries to include a repetition index, which is incremented for each repeat of each @ID, as shown below:

Here is a simple example with repeated nested fill-in Questions (QR):

QR1: Specify Location of each Additional Margin \_\_\_\_\_\_\_\_\_\_\_

QR2: Distance of Tumor from Margin (mm) \_\_\_\_\_\_\_\_

The data for 3 repeats may be stored with repetition indices (1, 2, 3) as follows:

|  |  |  |  |
| --- | --- | --- | --- |
| Q @ID | ListItem @ID\* | Response | Repetition Index |
| [QR1 @ID] | NULL | “3 o’clock, with suture at 12 o’clock” | 1 |
| [QR2 @ID] | NULL | “0.3” | 1 |
| [QR1 @ID] | NULL | “9 o’clock, with suture at 12 o’clock” | 2 |
| [QR2 @ID] | NULL | “2.0” | 2 |
| [QR1 @ID] | NULL | “Deep margin” | 3 |
| [QR2 @ID] | NULL | “1” | 3 |

\*In this example, no ListItem (LI) @IDs are available for storage, since QR items are used.

### Other Uses for @IDs

Although not exposed in the current FDF release, @IDs provide the eCC team with the ability to map template items from one template version to items present in a previous template version. @IDs also provide the means to map each template item to Common Data Elements (CDEs) and primitive “base” item types (e.g., a base QAS of type “Lymphovascular Invasion”), which resides in the eCC Base Template. This system has the potential to allow cross version and cross template queries.

Nevertheless, we plan to provide similar functionality through the future mapping of @IDs to external coding systems as noted above. These maps will be released as separate XML files, and will enable semantic-code-based queries, in addition to @ID-based queries.

Through these multi-faceted approaches, the CAP eCC team is committed to making the data representations as useful as possible for clinical care, cancer registry work and cancer research.

### Data Transmission

Transmission of eCC data must convey the essential information in the eCC model. NAACCR has created a very capable transmission model using HL7 2.51 format for cancer registry eCC data transmission. This is the recommended eCC transmission standard at this time. Much more information can be found in the [NAACCR Vol. V documentation](https://www.naaccr.org/pathology-laboratory-electronic-reporting/).

# eCC Reference Implementation using HTML



## Automatic DEF Generation

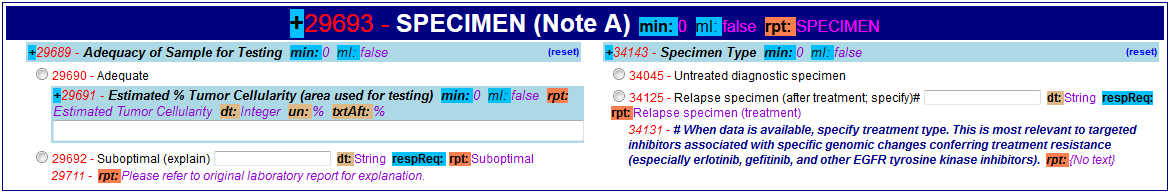
Although it is possible to create custom static DEFs (e.g., web pages, Windows forms) for each FDF, this is not required.  The FDF files can also be used for metadata-driven creation of DEFs via “just-in-time” DEF generation.

A sample XML🡪HTML DEF generator (using XSLT and CSS style sheets) is provided with each release, in the FDF folder. **The generated HTML DEFs are NOT intended to be functional eCC applications**; they are only guides to a possible implementation technique. Functionality, such as IIA, SDAC and SDS, has not been implemented.

The FDF documents may be viewed as HTML by dragging them into an xslt-compliant web browser (e.g., Firefox or IE). The eCC HTML pages contain toggle switches in the upper left corner for toggling the visibility of hidden metadata, IDs (@IDs), and [deprecated](#_Deprecated_Content_1) items.



Hidden metadata is displayed with colored icons as shown in the following image:



The icons from the previous figure are defined in the table below. Please note that the icon colors are subject to change. The table also shows the item types where the various elements and attributes may appear. The table illustrates how many attributes are used across different enh XML elements. Markings in red text have been deprecated, but may still exist in past or current enh XML templates.

### Table of commonly-used elements and attributes:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| # | enh XML | enh | Icon Appearance | Default Value | QS | QM | QR | LI | LIR | S | DI | LN |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| # | @mustImplement | \* | mI | true | x | x | x | x | x | x | x | x |
|  | <title> |  |  |  | x | x | x | x | x | x |  | x |
|  | <text> |  |  |  |  |  |  |  |  |  | x |  |
|  | @sort-order |  |  |  | x | x | x | x | x | x | x | x |
|  | @reportText | \* | rpt | null | x | x | x | x | x | x | x | x |
| # | @readOnly |  | locked | false | x | x | x | x | x | x | x |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| # | @alt-text |  | alt | “” | x | x | x |  |  |  |  |  |
| # | @maxCard | \* | max | 1 | x | x | x |  |  | x |  |  |
|  | @minCard | \* | min | 1 | x | x | x |  |  | x |  |  |
|  | @minCard = “0” |  | + |  | x | x | x |  |  | x |  |  |
|  | <required> |  |  |  | x | x | x |  |  | x |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| # | @responseRequired | \* | respReq | false |  |  |  |  | x |  |  |  |
| # | @omitWhenSelected | \* | omitWhenSel | false |  |  |  | x | x |  |  |  |
| # | @selection-disables-siblings |  | sdc | false |  |  |  | x | x |  |  |  |
| # | @selection-deselects-siblings |  | sds | false |  |  |  | x | x |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| # | @datatype |  | dt | “” |  |  | x |  | x |  |  |  |
|  | @textAfterResponse | \* | txtAft | “” |  |  | x |  | x |  |  |  |
|  | ResponseUnits |  | un | “” |  |  | x |  | x |  |  |  |

**#** Most icons are only displayed when the indicated attribute value is not the default value. These are indicated with “#” in the # column.

**\*** An asterisk in the enh column indicates an attribute that newly added for enh XML.

A separate user guide for the eCC XML Comparison Tool explains the icons for eCC metadata changes between versions. This tool and its icons are not described in this document

# Identifier (ID) FAQ

1. **What is a @ID?**
   1. The eCC XML files utilize an identifier format called the Composite Key (“@ID”) to identify each template item in each template. @ID identifiers are created in a decimal format that consists of a unique integer followed by a decimal point, and then a namespace integer which identifies the originating organization. An example of an identifier in @ID format is 1234.1000043, where 1234 is a non-repeating (unique) integer assigned from a pre-existing list of unique integers, and 1000043 is the namespace for the CAP.
      1. Identifiers using the @ID format are designed to unambiguously identify distinct items such as template versions (the **@template-id** attribute in eCC XML files), as well as individual items in each template (“item @IDs”). @IDs were developed to allow robust database management of identifiers for templates, Questions, and ListItems in data repositories. Unique @ID identifiers may be mapped to standard reference terminologies such as SNOMED CT, ICD-O3 and LOINC, and may also be used in data transmission protocols such as HL7 messages.
      2. As described later, the use of unique identifiers allows the eCC templates and line items to undergo versioning cycles with preservation of querying functionality.
   2. Note that the @IDs for identifying different eCC template versions (**@template-id**) are independent of the @IDs used for identifying FDF line items (item @IDs).
2. **Which types of template items in the eCC XML have @IDs?**
   1. All eCC **template** items, including Section, DisplayedItem, Questions and ListItem, are associated with unique ids, which are distributed in the @ID format.
3. **Do @IDs ever repeat?**
   1. [@IDs for template items](#_Identifiers) (item @IDs) never repeat among all template items.
   2. @IDs for template versions (**@template-id**) never repeat within the list of all templates versions.
   3. However, it is possible for a **@template-id** value to be the same as a template item @ID, since versioned templates (**@template-id**) and template line items (item @IDs) are fundamentally different types of objects. This is similar to unique identifiers in two different database tables (one for template versions and one for template items).
   4. Identifiers in @ID format derive from database columns that are associated with a “Unique Key” constraint. The eCC database therefore enforces the uniqueness of these identifiers
   5. Once a @ID is released, it cannot be reused.
4. **Why aren’t @IDs always in sequential order?**
   1. @IDs are derived from a pre-existing list of sequential integers that represent unused @IDs. Sometimes during the initial phases of template editing, blocks of template items are created, only to be rejected and then deleted before release. The unused @IDs from the deleted records may be reused, resulting in @IDs that are out of sequence, but still unique. In other cases, template items (or entire blocks of items) may be moved to different locations within the same template. (@IDs are never moved to other templates.) Thus, @IDs are not strictly maintained in sequential order.
5. **Do I need to use the @IDs on Sections and DisplayedItems?**
   1. @IDs for Sections and DisplayedItems are unique references to identify line items in the template XML. They are used to identify these structural elements in DEF implementations. @IDs for Sections and DisplayedItems are not needed for storing ListItem.
6. **Why are some <title>** **elements in the XML blank?**
   1. The visual output of the eCC is designed to be as close to the original case summary as possible. However, the eCC informatics model relies on storing both Question and ListItem codes for every user response stored in the data persistence layer. This model is commonly known as the Entity-Attribute-Value (EAV) model. In some cases, the EAV model required the creation of @IDs for “invisible” Questions. In the EAV model, Q @IDs are Attributes, and A @IDs are values. QR and LIR fill-in values are stored separately (e.g., in a separate database column).
   2. The invisible Questions have a special attribute in the XML named @alt-text. This attribute contains some text that can act as a readable substitute for the invisible Question text in the DEF. In many cases, the alt-text simply repeats the parent Question text, because the invisible Question functions to add specificity to the parent Question, but does not introduce new semantics.
      1. Invisible Questions may receive terminology codes in the future to aid reporting and querying. However, the Question **<title>** value will remain invisible to match the original presentation in the CCP version of the case summary.
   3. The values in the @alt-text attributes are subject to review and may change or be removed in the future. They are designed as simplified Question references and possible suggestions for explicit DEF display text. *In many cases, display of the alt-text may* ***decrease*** *readability and user acceptance. The alt-text terms are* ***not*** *designed for use in clinical reports.*
7. **Do @IDs change when a new version of a template is released?**
   1. If the changes to a template are minor (e.g., spelling changes, deletion of blank lines, minor rewording), then the @ID identifiers will remain unchanged.
   2. If a template undergoes a major version update, then a new @ID identifier could be generated for each line of each new template version. However, if the changes are not extensive, @IDs will be preserved where possible.
8. **If the @IDs can change between versions, how can we combine data from multiple versions?**
   1. **Before attempting cross-version queries, it is important to ensure that the ListItems for the Question item are essentially unchanged across all the versions**. Blindly querying across different versions of a Question, when the different versions may have different ListItem sets, can return incorrect results in some cases. When ListItem lists for a Question change, then the Question (and its @ID identifier) is deprecated and replaced. ListItems in a QAS can change as well. If the data analyst is aware of the change implications, it should be possible to combine similar QAS blocks into a common query and result dataset. In these cases, interpretation of the query results requires that the changes over time be considered.

# Summary

This SDC Primer provided a brief introduction to the major SDC form design features. It provided a general introduction to the “philosophy” of the IHE SDC model. It then addressed some details for the most important SDC features and a few special topics such as form validation. Although the core of SDC has stabilized over the past 4 years, existing features continue to be refined, and more features (e.g., form behavior rules) are being added.

SDC documentation will improve over time. For now, more information, Schemas, sample FDFs and sample applications can be found in the [October 2016 IHE SDC Profile](http://ihe.net/uploadedFiles/Documents/QRPH/IHE_QRPH_Suppl_SDC.pdf) and at the [SDC GitHub site](https://github.com/IHE-SDC-WG). Detailed descriptions of SDC elements and attributes are contained within the SDC Schema files, in the documentation tags. For assistance with SDC implementations or related Questions, please contact the Structured Data Team at the College of American Pathologists: call (847) 832-7700 or email [capecc@cap.org](mailto:capecc@cap.org).

# Glossary

## Terms and Abbreviations

**ChildItems**: The ChildItems element under a S or Q element provides a wrapper to subsume child XFCs including Q, S, DisplayedItem, ButtonAction and InjectForm. Note that a LI may not be a direct child of ChildItems, even though LI is an XFC. The ChildItems wrapper element serves to separate child XFC elements from other parent-owned metadata elements that supplement the parent XFC.

**Common Data Element (CDE):** When Data Elements (DEs) are found inside a DE repository/registry, a DE is called a CDE. A **CDE registry** allows the sharing of DEs among many forms and allows DE variations and version history to be maintained in a central location.

**Composite globally-unique identifier (CGUI):** When referring to a specific XFC in a specific FDF, a CGUI is creating by using the @baseURI of the closest ancestor, concatenated as a prefix with the @ID value of the XFC.

**Conditionally Required (CR)**: All required DEs (@mustImplement = true) must appear in DEF so that users may complete them. However, the responses from some required DEs should be **omitted** from **reports** under specific circumstances. For example, if no lymph nodes are present in a pathology specimen, then "required" lymph node pathology DEs should not be reported. These required DEs can be considered as **"Conditionally Required" (CR),** so that a user (or computer) may recognize when the DE should be omitted from the report. There are 3 basic types of CR: Omit When Selected (OWS), Omit When Unanswered (OWU) and Selection Disables Children (SDAC).

**Context** refers to the hierarchical relationship between form components defined by the FDF XML. Context (especially ancestors) can greatly affect the semantics of the DE units in an FDF and DEF. Incorrectly separating the DEF user responses into separate data slots (e.g., "shredding" captured QAS responses into database tables and fields not designed for SDC) can result in loss of context and misinterpretation of. Care must be taken to assure the preservation of the original context, including when storing DEF responses and assigning terminologies/codes. Preservation of context is a major advantage of SDC over other technologies. Because of the complexities of finding suitable terminology and CDE codes that precisely describe the contextual semantics of a captured DEF user response, application of terminology codes is not a substitute for SDC context preservation.

**Control**: XFCs can be rendered in a DEF in different ways depending on a Form Filler’s programming technique and visual themes/preferences, but all SDC forms use the same SDC information model and form components. When XFCs are rendered in a DEF, the DEF screen objects are called *controls* or *widgets* to distinguish these rendered screen items from the XFCs described by the FDF XML.

**Data Element (DE)**: A DE is essentially a Q, along with a definition of acceptable answer choices (either a direct "fill-in" response or a list of possible answer choices).DEs are a fundamental concept in the SDC world. DEs can live in multiple environments, including, e.g., inside a text document, XML, public DE registry, FDF, DEF, or database. In some cases, terminology codes and many other types of metadata may be added to DEs. When multiple DE units are nested together into an interdependent unit, the structure is known as a **complex DE**. Many FDFs contain complex DE blocks. Although there is no universal XML standard for representing and transmitting DE structure and content, FDF XML can be used for this purpose.

**Data items** live inside DEFs. Data items are simply Qs that need an answer. The Data Element (DE) and the Question/Answer Set (QAS) are closely related concepts.

**Data-Entry Forms (DEFs)** can be found in web pages, desktop software or any data-entry screen in any software application. **Data items** live inside DEFs. Data items are simply Qs that need an answer. Structured Data Capture (SDC) is a new technology that creates interoperable, computer-readable definitions for standardized **data items** in DEFs.

**DemogFormDesign** is an element in an **SDCPackage** that is used to contain a generic demographic form that is reused with multiple different domain-specific forms that are placed in the FormDesign S.

**DisplayedItem (DI):** The **DI** element is used to define visible text for display almost anywhere in a DEF. Each DI has a unique @ID, but, unlike Qs and Ss, a DI cannot have a ChildItems element or XFC descendants. Displayed text that must have XFC descendants must use the S element instead; this should not cause a problem because the formatting of any SDC element, such as DI or S, should be determined by the @styleClass attribute, not the element type.

**FDF-R** is an FDF containing captured user responses.

**Form Creators** are the most important component of the SDC ecosystem, despite not being IHE **actors** (transaction nodes). Form creators are individuals and organizations that define the content, structure and behavior of FDFs. Form Creators may develop FDFs by working directly with FDF XML, or by using a tool to create/edit FDF files.

**Form Design File (FDF)**: SDC defines the data items of a DEF inside a Form Design File (FDF). An FDF is primarily an XML description of the data items in a DEF. It is not dependent on the programming language used to create a DEF.

**Form Managers (FMs)** store FDFs in a repository and transmit them immediately in response to requests from Form Fillers (FFs). Form Creators send (e.g., via email or direct upload) FDFs to one or more Form Managers.

**Form Receivers (FRs)** receive the SDC response data (in an FDF-R file) from the **Form Filler** and process the data according to the FR’s needs. FRs are responsible for storing the captured SDC data as native SDC XML and/or "shredded" into some other storage format.

**FormDesign** is the root element of an FDF. FormDesign contains an @ID attribute and several other attributes specific to the FDF root. In addition, FormDesign may contain any number of standard or custom Property elements to provide essential metadata about the form.

**Implied Activation** means that child items are automatically activated when a user response is captured in a parent QAS. Thus, when a LI is selected, all child items of the LI become activated. QS, QM and QR items also may have direct child items in the FDF and DEF. For QM and QS, the child items are activated when any LI is selected. For QR, the child items are activated when a user response is captured in control.

i**nvisible Questions** have no @title value in the XML or visible text in the DEF, but their LIs do have @title values and thus text in the DEF.

**ListItem (LI):** LIs are answer choices that are subsumed by a QS or QM. An LI comes in two basic types: the **simple LI**, which is selectable by the user, and the **ListItem-Response** (**LIR**), which is selectable and can also capture a user-entered response.

**ListItem-Response (LIR)**: An LIR is a LI that contains a Response element for a user to fill in a response. If the LIR is selected by a user, then the user can also add a strongly-typed response to the fill-in portion of the LIR.

**ListItemResponseField** is the part of an LIR that receives and contains the user’s response data. It contains a strongly-typed data entry element for restricting and validating the type of data that may be entered.

**Metadata** refers to the values of all FDF attributes, which affect the rendering, behavior, storage, exchange, reporting and interpretation of all FDF form components. In SDC, everything inside an FDF is metadata except for the *data* entered by the user and captured by the DEF.

**Multi-select Question (QM):** A Q that allows the user to select multiple LIs and/or LIRs in a QAS list.

**Notes** are strings of text that may accompany DE controls to provide explanatory information with background documentation. Notes usually derive from the SDC **DisplayedItem** element.

**Property**: Properties may be used to define standard or custom FDF metadata. Properties can be used to display static information on forms, and to display context-sensitive information (e.g. tooltips, help pop-ups, or special report text). They also may be used to record visible or hidden metadata for any purpose. Properties may be nested to any depth and use any datatype for the Property’s value.

**Question (Q)**: Qs that can capture a response (fill-in) value directly are called **Question-Response Items (QR).** They are sometimes also called Q Response (QR) items. QR response metadata and captured response values are handled by the **ResponseField**\**Response** structure. Other Qs take a List of answer choices, called **ListItem**s (**LI**). In the most common cases, Qs with LI choices are either **QS** or **QM**. QS and QM items never have a Response element, however, selected LIs may accept a user response in the **ListItemResponseField**.

**Question/Answer Set (QAS)**: A QAS is a Q in a DEF that includes a definition of its permitted answer(s). The permitted answers may be ***captured*** as a filled-in **response** by the user, or by ***selecting*** from a list of answer choices (**LI** elements). A QAS can be thought of as a DE that lives in a DEF rather than in an FDF or CDE registry.

**Question-Response Items (QR)**: Qs that can capture a response (fill-in) value directly are called **Question-Response Items (QR).** They are sometimes also called Q Response (QR) items. A QR contains no child LIs. Response metadata and captured response values are handled by the **ResponseField**\**Response** XML structure.

**SDCPackage** is an XML container sent from a Form Manager to a Form Filler. It may contain an **HTMLPackage** enclosing an HTML form, or it may contain an **XMLPackage** that encloses a FormDesign S and optionally, a DemogFormDesign S. The HTML package contains a ready-to-use HTML SDC form. The XML Package must be converted into an implemented DEF and then rendered to the computer screen. For example, the XMLPackage is converted to an HTML DEF inside the Form Filler.

**Section (S):** The most basic structure of an FDF is the **S**. Three specially-named Ss are the **Section**, **Body** and **Footer**, which appear as direct children of the top-level FormDesign element. Ss can contain a variety of metadata features, but their main function is to group XFCs into logical groupings.

**Single-select Question (QS)**: A Q that allows the user to select only one LI from a List of LIs.

**Structured Data Capture (SDC)** is a new technology that creates interoperable, computer-readable definitions for standardized **data items** in DEFs.

**XML Form Component (XFC)**: The parts of FDF XML used to represent primary form objects are called *XML Form Components.* XFCs may be one of several types: Q, LI, S, DI, InjectForm, and ButtonAction. All XFCs have a required @ID attribute. Q, S and LI may subsume a ChildItems element, which contain other XFCs.

## Important SDC Attributes

[O]=Optional in SDC XML, [R]=Required in SDC XML; a data type for each attribute is also provided in square brackets.

(XML attributes that have a default value need not appear in the XML. If these attributes are missing from the SDC XML, they are treated in the same way as if they were included in the XML with their default value. Default values on attributes are listed below.)

**@basedOnURI** [O] [URI]**:** Appears on FormDesign and DemogFormDesign. URI that identifies the previous FDF that that the current FDF is based upon. In most cases, this should be a standard SDC FDF that is modified and/or extended by the current FDF.

**@baseURI** [O] [URI]: Appears on all XFCs. An institutionally-unique string, ideally based on a registered domain name, that acts as the namespace for the @ID values in an FDF. Concatenating @baseURI and @ID yield a composite globally-unique identifier (CGUI).

**@enabled**: [O] [boolean] [default=true]: Appears on all XFCs. Determines whether the user can interact with a DEF control.

**@filename** [O] [string]: Appears on FormDesign and DemogFormDesign. The filename of the FDF when is saved to a file storage device (e.g., a disk or USB drive). The @filename appears inside the FDF XML to help ensure the identity of the FDF content in case the saved filename (on a disk drive, etc.) has been changed or any reason.

**@formTitle** [O] [string]**:** Human readable title for display when choosing forms from list provided by a FM.

**@formInstanceURI** [O] [URI]: Appears on FormDesign and DemogFormDesign. Globally-unique URI used to identify a unique instance of an FDF, when displayed as a DEF. Used for tracking form responses across time and across multiple episodes of editing by end-users. This URI does not change for each edit session of a form instance.

**@formInstanceVersionURI** [O] [URI]: Appears on FormDesign and DemogFormDesign. Globally-unique URI used to identify a unique instance of a form. Used for tracking form responses across time and across multiple episodes of editing by end-users. This URI must change for each edit session of a form instance.

**@formPreviousInstanceVersionURI** [O] [URI]: Appears on FormDesign and DemogFormDesign. Unique URI used to identify the immediate previous instance of a form containing responses. This is the @formInstanceVersionURI that represents the instance of the form that the user opened before beginning a new cycle of edit/save. This attribute is used for tracking form responses across time and across multiple episodes of editing by end-users. The URI value must change for each edit session of a form instance.

**@formTitle** [O] [string]: Appears on FormDesign and DemogFormDesign. Human readable title for display when choosing forms from a list.

**@maxCard** [O] [integer] [default=1]: Appears on S and Q. The maximum number of repetitions allowed for a S or Q. A value of 1, indicates that the S or Q cannot be repeated on the data entry form. A value of 0 indicates that the number of repeats is unlimited. If not 0, then it must be greater than or equal to @minCard.

**@maxSelections** [O] [integer] [default=1]: Appears on ListField. Maximum number of answer choices (ListItems) that can be selected by the user. If set to 0 or >1, then Q is a QM, otherwise it’s a QS.

**@minCard** [O] [integer] [default=1]: Appears on S and Q. The minimum number of repetitions allowed for a S or Q. A value of 1 indicates that the user must answer any Q(s) that has this attribute set. If *@minCard* is set to 0, then the item and all descendent Qs are optional to answer.

**@minSelections** [O] [integer] [default=1]: Appears on ListField. Minimum number of QM answer choices (ListItems) that must be selected by the user if the QM is being answered.

**@mustImplement** [O] [boolean] [default=true]: Appears on all XFCs. If set to "true", then the form implementation must make this item available for use on the form. If this attribute is set to "false", then the item and all descendants do not need to be available in the DEF

**@name** [O] [string]: Appears on all elements. Developer assigned identifier, similar to a unique component/object name, used to provide a variable name for programmatic manipulation of an element.

**@omitWhenSelected**: [O] [boolean] [default=false]. Appears on LI. If OWS set to true, then the Q and its response(s) should not be present in a typical report derived from this template. This attribute can be set to true when the ListItem is used e.g., to (1) control form behavior (e.g., a ListItem that, when selected, activates a set of child Qs), or (2) when the Q provides unhelpful "negative" information about actions that did not occur or were not performed, or (3) when an ListItem selection describes things that were not observed or could not be assessed, and thus can be omitted form the report. If this attribute is set to false (the default) then the Q and its response(s) should appear in the report.

**@prevVersionID:** [O] (URI) Appears on FormDesign and DemogFormDesign. Identifies the SDC form that is the immediate previous version of the current FDF. This is a previous form version, not the version of the saved responses (referred to by @formInstanceVersionURI). This attribute may be used to assemble a linked list of FDF versions and allows FDF comparison tools to compare a form version with the immediate past version. Walking back along the linked list of FDF versions with an FDF comparison tool allows a user to observe the complete history of XML changes to the FDF XML.

**@readOnly** [O] [boolean] [default=false]**:** Appears on S and Q. If set to true, the S or Q should not be editable by the user even if appears to be enabled and visible in the DEF. Any default values set in FDF should be considered part of the stored data set responses and be transmitted to a Form Receiver.

**@responseRequired** [O] [boolean] [default=false]: Appears only on LI ResponseField, for LIs that can capture a response (LIR). If set to true, then the appropriate user response must be entered in the @val attribute for that field.

**@selectionDeselectsSiblings** (SDS) [O] [boolean] [default=false]: Appears on LI. If set to true and @minCard <> 1, then selecting a LI control de-selects all its sibling LIs controls and ignores any user-entered data.

**@selectionDisablesChildren** (SDAC) [O] [boolean] [default=false]: Appears on LI. If SDAC is set to true on a QM LI, then selecting the LI control disables all its descendant controls and ignores any user-entered data. Deselecting the component re-enables the child components.

**@title** [O] [string]: Appears on all XFCs. @title is the primary text to show on the form’s components. Also known a s "prompt" or "label" or "visible text" or "caption"

**@val** [O] [string]: Appears on all elements that can accept a value, including all data-typed elements, Property and Comment. An alphanumeric string attribute that defines a response value or a pre-coded metadata value. The datatype of @val is set to an SDC-supported data type.

**@visible** [O] [boolean] [default=true]: Appears on all XFCs. Indicates whether a control can be seen on a form.

# Data Types

## Numeric Data Types (reference)

Note that these data types derive from the Decimal data type (except for decimal itself)

|  |  |
| --- | --- |
| ***Name*** | ***Description*** |
| byte | A signed 8-bit integer |
| decimal | A decimal value |
| int | A signed 32-bit integer |
| integer | An integer value |
| long | A signed 64-bit integer |
| negativeInteger | An integer containing only negative values (..,-2,-1) |
| nonNegativeInteger | An integer containing only non-negative values (0,1,2,..) |
| nonPositiveInteger | An integer containing only non-positive values (..,-2,-1,0) |
| positiveInteger | An integer containing only positive values (1,2,..) |
| short | A signed 16-bit integer |
| unsignedLong | An unsigned 64-bit integer |
| unsignedInt | An unsigned 32-bit integer |
| unsignedShort | An unsigned 16-bit integer |
| unsignedByte | An unsigned 8-bit integer |
| Restrictions on Numeric Data Types Restrictions that can be used with Numeric data types | |
| @enumeration |  |
| @fractionDigits |  |
| @maxExclusive |  |
| @maxInclusive |  |
| @minExclusive |  |
| @minInclusive |  |
| @pattern |  |
| @totalDigits |  |
| whiteSpace |  |

## String Data Types (reference)

|  |  |
| --- | --- |
| Note that all the data types below derive from the String data type (except for string itself)! | |
| ***Name*** | ***Description*** |
| ENTITIES |  |
| ENTITY |  |
| **@ID:** | A string that represents the @ID attribute in XML (only used with schema attributes); **it provides a unique identifier that is required for most controls** |
| IDREF | A string that represents the IDREF attribute in XML (only used with schema attributes) |
| IDREFS |  |
| @language | A string that contains a valid language id |
| @name | A string that contains a valid XML name |
| NCName |  |
| NMTOKEN | A string that represents the NMTOKEN attribute in XML (only used with schema attributes) |
| NMTOKENS |  |
| normalizedString | A string that does not contain line feeds, carriage returns, or tabs |
| QName |  |
| string | A string |
| token | A string that does not contain line feeds, carriage returns, tabs, leading or trailing spaces, or multiple spaces |
| Restrictions on String Data Types | |
| Restrictions that can be used with String data types: | |
| @enumeration | |
| @length | |
| @maxLength | |
| @minLength | |
| @pattern (NMTOKENS, IDREFS, and ENTITIES cannot use this constraint) | |
| @whiteSpace | |

## Date and Time Data Types (reference)

|  |  |
| --- | --- |
| ***Name*** | ***Description*** |
| date | Defines a date value |
| dateTime | Defines a date and time value |
| duration | Defines a time interval |
| gDay | Defines a part of a date - the day (DD) |
| gMonth | Defines a part of a date - the month (MM) |
| gMonthDay | Defines a part of a date - the month and day (MM-DD) |
| gYear | Defines a part of a date - the year (YYYY) |
| gYearMonth | Defines a part of a date - the year and month (YYYY-MM) |
| time | Defines a time value |
| Name | A string that contains a valid XML name |
| NCName |  |
| NMTOKEN | A string that represents the NMTOKEN attribute in XML (only used with schema attributes) |
| NMTOKENS |  |
| normalizedString | A string that does not contain line feeds, carriage returns, or tabs |
| QName |  |
| string | A string |
| token | A string that does not contain line feeds, carriage returns, tabs, leading or trailing spaces, or multiple spaces |
| Miscellaneous Data Types (reference) | | |
| ***Name*** | | |
| anyURI | | |
| base64Binary | | |
| boolean | | |
| double | | |
| float | | |
| hexBinary | | |
| NOTATION | | |
| QName | | |
| Restrictions on Miscellaneous Data Types | | |
| Restrictions that can be used with the other data types: | | |
| enumeration (a Boolean data type cannot use this constraint) | | |
| @length | | |
| @maxLength | | |
| @minLength | | |
| @pattern (NMTOKENS, IDREFS, and ENTITIES cannot use this constraint) | | |
| @whiteSpace | | |

1. The format for the FDF XML file is defined by an XML Schema. The FDF XML defines the information (e.g., Questions and answer choices) that must be displayed in a computer screen (DEF), and also describes essential features of the DEF behavior when the user is interacting with the form on a computer screen. [↑](#footnote-ref-2)
2. A CDE is a DE designed for widespread use and is housed in a CDE registry. [↑](#footnote-ref-3)
3. In many cases, CDEs and terminologies (e.g., SNOMED CT) are insufficient to describe the full nuanced meaning of a captured response in a DEF and consideration of the DEF context is required to fully understand the captured user responses. However, annotation of FDF-DEs with CDEs and terminologies is useful in many cases, such as when aggregating data from SDC DEFs and other sources. [↑](#footnote-ref-4)
4. i.e., back and forth exchange of one SDC dataset (the data from an SDC DEF, inside an FDF-R) across multiple nodes, with no loss of data or context. The original DEF and its user-entered data will be visible and unchanged regardless of how many times it has been transmitted across multiple nodes. [↑](#footnote-ref-5)
5. This allows the use of complex rich text (e.g., fonts, special text formatting) alongside the equivalent plain (unformatted) text in FDFs. Although many DEFs appear “better” with rich text, some systems cannot support it. Also, many data transmission standards do not natively support rich text of any sort. [↑](#footnote-ref-6)
6. FMs actually store FDFs inside SDC Packages, described later. [↑](#footnote-ref-7)
7. In the case of HTML forms, the FF programming code (e.g., JavaScript) that controls the DEF behavior runs inside the HTML page and the web browser. [↑](#footnote-ref-8)
8. It is also possible to transmit data to FRs using other transport formats such as NAACCR Vol V (an HL7 2.51 format), but alternate transmission formats require more effort and are out of scope for this document. [↑](#footnote-ref-9)
9. “Weakly-defined” in this context means that the domain and range of possible answers are usually not enumerated or precisely defined. [↑](#footnote-ref-10)
10. Responses (e.g., selected answer choices, or typed-in data) that are entered into a DEF by a user are said to be “captured” as data by the DEF. The computer will insert this captured data directly into the FDF in standard locations in the XML. The captured data inside the FDF can be sent (transmitted) to other locations, such as different nodes in a computer network. [↑](#footnote-ref-11)
11. Context refers to the hierarchical relationship between form components defined by the FDF XML. Contextual semantics refers to the understanding or meaning of a Question, combined with the captured answer, as affected by their position in the DEF hierarchy of QAS items. Syntax refers to the standardized structure of data items inside the FDF-R, as defined by the SDC XML Schema. [↑](#footnote-ref-12)
12. Note that the letters “DE” in “DEF” do **not** stand for *data element.* DEF stands for *data-entry* form. Also, the word “Element” in “Data Element” has no intended or intrinsic relationship to an XML “element.” [↑](#footnote-ref-13)
13. For our purposes, an information model is a logical design, expressed e.g., as an XML Schema or UML diagram, that is used to model generic information, rather than specific domains of information (e.g., medications or car parts). The SDC information model uses the generic DE paradigm to represent any information domain. [↑](#footnote-ref-14)
14. Reference to formal SDC components will henceforth be represented with capitalized terms, e.g., Question, Section, etc. [↑](#footnote-ref-15)
15. The terms “ListItem” and “answer choice” are used interchangeably in this document. ListItem refers to the SDC element name for an answer choice in the DEF. [↑](#footnote-ref-16)
16. In the SDC model, each ID must be unique inside its form. However, it’s possible for the exact same ID to be found in a different form, identifying a completely different object; this repeat use of IDs is strongly discouraged, but may be required in some legacy use cases. When a form is versioned, e.g., from version 1 to version 2, the IDs of the XFCs are generally preserved, unless the XFC changes significantly. This process, including the determination of significant changes that force an ID change in a versioned form, will be discussed in detail later. [↑](#footnote-ref-17)
17. It is possible to bypass this restriction when using an SDC XML or HTML element, or when using a custom SDC Extension. Use of these exceptions requires the declaration of an external namespace and an XML Schema, e.g., for using XHTML to record richly-formatted text inside an FDF. [↑](#footnote-ref-18)
18. This applies to both data and metadata, and also applies to all SDC datatype elements which follow W3C conventions. [↑](#footnote-ref-19)
19. Several required attributes have been omitted for this example [↑](#footnote-ref-20)
20. “Sticks” means that the Header and Footer do not scroll off the page when the user scrolls through the Body Section of the SDC DEF. [↑](#footnote-ref-21)
21. In addition to the displayed FormDesign parts, the FormDesign element may contain custom Extensions, Comments, and Events (BeforeLoadForm, BeforeLoadData, BeforeShowForm, BeforeDataSubmit, BeforeCloseForm and OnEvent). These will be covered later. [↑](#footnote-ref-22)
22. The attributes directly under FormDesign are described [later](#_FormDesign_Metadata_Declarations). [↑](#footnote-ref-23)
23. A DI, like all XFCs, may also display rich content such as images, links or video. [↑](#footnote-ref-24)
24. @styleClass is available on nearly all SDC elements. It is intended to assist with formatting if the element is displayed in a DEF. [↑](#footnote-ref-25)
25. ListItems can be handled as single-select or multi-select. This will be discussed in a later section. [↑](#footnote-ref-26)
26. Other uses for ListField and its substructure will be explained later. [↑](#footnote-ref-27)
27. The uniqueness of the baseURI value can be guaranteed by using a namespace registry such as the Internet DNS system, administered by [ICANN](https://en.wikipedia.org/wiki/ICANN) and [IANA](https://en.wikipedia.org/wiki/Internet_Assigned_Numbers_Authority). [↑](#footnote-ref-28)
28. In the example above, the dereferenced CGUI, [https:// [www.cap.org/FormsDomainSDC1/100](http://www.cap.org/FormsDomainSDC1/100)](https://www.cap.org/MyForms123/100), could be typed into a web browser to obtain information about that XFC, and possibly information about the specific form (MyForms123), and the group or author that created it. However, the ability to dereference a baseURI is determined by the use case. [↑](#footnote-ref-29)
29. camelCase is a format that starts with a lower-case letter and uses capitals at the start of new words. Spaces are not permitted. [↑](#footnote-ref-30)
30. For our purposes, the classes of an object-oriented languages are equivalent to the Types of an XML Schema. [↑](#footnote-ref-31)
31. XFC Types are the SDC Schema Types that define the XFC elements in the FDF. Each Schema Type ends with the suffix “Type” and is otherwise similar in name to the XFC XML element that it defines. [↑](#footnote-ref-32)
32. For example, ButtonItemType, ListItemType and RepeatingType are siblings, in that each inherits from the parent Displayed Type. [↑](#footnote-ref-33)
33. The XML element “Section” is defined by **SectionItemType** in the SDC Schema. [↑](#footnote-ref-34)
34. **DisplayedType**, which defines the DisplayedItem element, is not listed again here because it is a parent type, and therefore its elements and attributes were listed in Example 8. [↑](#footnote-ref-35)
35. The W3C ID type, derived from NCName, is not the same as the SDC ID attribute. The SDC ID is derived from the W3C URI type, which is much more permissive regarding the text format. [↑](#footnote-ref-36)
36. Source: [www.datypic.com/sc/xsd/t-xsd\_NMTOKEN.html](http://www.datypic.com/sc/xsd/t-xsd_NMTOKEN.html) [↑](#footnote-ref-37)
37. Direct derivatives of datatypes generally extend a datatype by adding one or more attributes. The derived datatypes are used to build up more complex Types, which *are* derivatives of ExtensionBaseType. This makes it possible to add Property, Comment and Extension elements to all derived SDC Schema Types and the elements defined by them. [↑](#footnote-ref-38)
38. This refers to all SDC elements that descend from ExtensionBaseType. [↑](#footnote-ref-39)
39. Property names, value enumerations and display behavior should be defined in use-case-specific profiles, FF software and style sheets. Enforcement of Property content and display behavior takes place, e.g., during the quality testing phase as part of the form modelling workflow. By design, the SDC Schema does not enforce Property element content beyond the basic Property structure defined by the SDC Schema. This allows generic Property customization for special use cases. [↑](#footnote-ref-40)
40. Attribute text and XHTML strings in SDC Properties should preserve whitespace when rendered in a DEF, and special characters that disrupt XML and processing must be escaped out using standard XML escaping rules. These rules apply to any text in any SDC attribute. [↑](#footnote-ref-41)
41. To ensure proper rendering in all FFs, formatted XHTML should generally always have equivalent non-formatted title text as well. XHTML text must use the XHTML namespace and Schema. The decision to display Property\HTML text depends on the rendering capabilities of the FF. [↑](#footnote-ref-42)
42. We could instead have declared the xhtml namespace at the top of the XML document (using xmlns:h="http://www.w3.org/1999/xhtml") and prefixed the xhtml elements to produce less verbose XML (e.g., using the “h” namespace prefix, **<h:div/>**) , but that could interfere with comprehension of this short example. [↑](#footnote-ref-43)
43. A Question XFC may have child ListItem XFCs and nested XFCs. Each of these XFCs contains its own unique @ID which must correspond to the @ID assigned to the matching DEF control. @ID content values may not repeat within an FDF. [↑](#footnote-ref-44)
44. The QR is sometimes called a Question-Fill-in (QF). [↑](#footnote-ref-45)
45. SDC supports W3C datatypes and also supports XML and HTML datatypes for user responses. [↑](#footnote-ref-46)
46. i.e., they begin with a lower-case letter, but each new word part is capitalized and appended to the first word part, e.g., “nonNegativeInteger”, “integer”, “gYear”, etc. Almost all SDC attributes use this format. The one notable exception, because of its importance, is @ID. [↑](#footnote-ref-47)
47. “xs” is a common namespace prefix for XML Schema, and refers to the fact that the datatypes are defined in the XML Schema documentation. [↑](#footnote-ref-48)
48. All default XML attribute values may be omitted from the XML, but should be treated as if they were present, holding the default value. [↑](#footnote-ref-49)
49. Some SDC implementations permit descendant answers to be selected (e.g., LI.2a) before ancestor answers (LI.1b) are selected. This requires that descendant Questions (**QS2**) are activated in the DEF before the user responds to ancestor Questions (**QS1**). In these cases, the ancestor answer (LI.1b) must be automatically-selected when a descendant Answer (LI.2a or LI.2b) is selected by the user. Note that ancestor QR and LIR responses (e.g., **QR3**) cannot be automatically filled out when descendants (LI.4a or LI.4b) are selected. There are other issues with this model as well. The programming for this approach is more difficult and error-prone and can result in inadvertent errors from improperly completed DEFs. Further discussion of this model is beyond the scope of this document. [↑](#footnote-ref-50)
50. *Active* means that a form control can be manipulated or answered by the user. Thus, an activated control is both *enabled* and *visible* to the user. An *inactive* control is *disabled* (grayed out) and may also be *invisible*. [↑](#footnote-ref-51)
51. CAP eCC Note: Currently, some CR Questions in a CCP may have a “+” prefix (indicating optional status, but the use of “+” is not completely consistent across CR items in all CCPs. Nevertheless, all eCC CR items *must* appear in the DEF. Appropriate omissions of CR items from the pathology report are compliant with CAP and CoC accreditation standards. [↑](#footnote-ref-52)
52. Note that minCard = ”0” will also cause a “+” prefix to be displayed on all OWU Questions in the SDC HTML (see Question 2 in the figure), and this “+” is a signal to users that the Question may be left blank if inapplicable. Despite the “+” prefix, all CR Questions *must be displayed* in the DEF, because in OWU cases, mustImplement **= “true”**, and they *must be answered* if the DEF-user determines that they are applicable. [↑](#footnote-ref-53)
53. There are ways around this “deselection” limitation of a QS, such as the use of a “reset” function as shown in the QS figures. However, most QM check box implementations support deselection natively. [↑](#footnote-ref-54)
54. By writing “required to answer,” we always assume that the QM is activated and reachable in the DEF’s QAS tree hierarchy. The meaning of “required’ depends on the use case, but in general, it means that the DEF will validate that the Questions has a response and inform the user if a response is missing. That validation is impossible when there is only a single checkbox, because the computer can’t distinguish an intentionally unchecked checkbox from a box that was accidentally skipped by the user. [↑](#footnote-ref-55)
55. The @readOnly status is not inherited by descendants of the locked Question or its ListItems. [↑](#footnote-ref-56)
56. SDAC was previously abbreviated as “SDC.” The abbreviation was changed to avoid conflict with the ONC (Office of the National Coordinator) Structured Data Capture (SDC) standard. [↑](#footnote-ref-57)
57. In these cases, the single LI is generally rendered as a check-box, not an option button, and the Question is treated as a QM. This is because a single checkbox can always be deselected, while a single option button cannot usually be deselected. [↑](#footnote-ref-58)
58. Questions are only applicable if they are activated and have been reached by the user as s/he navigates through the form Question hierarchy. [↑](#footnote-ref-59)
59. For example, Questions may be required for accreditation, clinical pathways, and/or quality assessment purposes. [↑](#footnote-ref-60)
60. The SDCPackage may contain an HTML package instead of an XML Package. However, this usage is out of scope for the Primer. It may additionally contain one Admin section, and one or more Submission Rule and/or Compliance Rule sections. These are also out of scope for this document. Interested readers should consult the SDC Schema and the SDC Profile documentation. [↑](#footnote-ref-61)
61. All SDC schemas are available at <https://github.com/IHE-SDC-WG> [↑](#footnote-ref-62)